Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Dean Gostiša

Visoko razpoložljiv tekmovalni sistem v oblaku za tekmovanje Bober - CaaS

diplomsko delo

univerzitetni študijski program računalništvo in informatika

Mentor: dr. Andrej Brodnik

Ljubljana 2014

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Tekmovanje Bober je tekmovanje, kjer v kratkem času tekmuje veliko število tekmovalcev. Poleg velikega števila hkratnih tekmovalcev so postavljene še relativno stroge zahteve glede zanesljivosti delovanja, saj mora sistem zagotavljati zelo majhno število izgubljenih sej.

V diplomski nalogi najprej oblikujte osnovne kvantitativne in kvalitativne kazalnike, katerim bi moral zadoščati razviti sistem. Nato preglejte druge podobne obstoječe tekmovalne sisteme in jih ovrednotite glede na oblikovane kazalnike. Iz tako ovrednotenih sistemov oblikujte funkcionalne zahteve za razvoj novega tekmovalnega sistema v oblaku (CaaS - Competition as a Service), ga arhitekturno porazdeljeno in povečkrateno zasnujte, razvijte ter praktično ovrednotite.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Dean Gostiša, z vpisno številko **63080050**, sem avtor diplomskega dela z naslovom:

Visoko razpoložljiv tekmovalni sistem v oblaku za tekmovanje Bober - CaaS

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom dr. Andreja Brodnika,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki „Dela FRI“.

V Ljubljani, dne 20. maj 2014 Podpis avtorja:

Zahvalil bi se mentorju, prof. dr. Andrej Brodniku, za strokovno pomoč in usmerjanje med izdelavo diplomskega dela. Prav tako bi se zahvalil vsem študentom, posebej Milutinu Spasiću in Lovrotu Podgoršku, s katerimi smo skupaj razvijali sistem ter ostalim, ki so mi stali ob strani vsa leta študija, predvsem svoji družini in punci Tatjani.

Kazalo

Seznam slik

Seznam tabel

Seznam primerov

Povzetek

Diplomsko delo opisuje implementacijo nove tekmovalne platforme za tekmovanje Bober, ki je mednarodno tekmovanje v računalniškem razmišljanju in pismenosti za osnovnošolce in srednješolce. Do sedaj smo v Sloveniji za izvedbo uporabljali sistem Majava, a smo ob povečanju števila tekmovalcev leta 2012 ugotovili, da sistem zaradi svoje implementacije in tehnične postavitve ni več primeren za izvajanje tekmovanj vnaprej.

Analizirali smo obstoječe delo na področju tekmovalnih platform, sestavili kvantitativne in kvalitativne zahteve delovanja, razvili podatkovni model, implementirali vse potrebne funkcionalnosti in poskrbeli za postavitev sistema. Nova platforma uporablja 3-slojno porazdeljeno arhitekturno zasnovo, ki nam omogoča uspešno porazdelitev vseh slojev na več strežnikov, s čimer smo zagotovili visoko razpoložljivost in povečljivost sistema.

Z novim sistemom smo v letu 2013/14 uspešno izvedli šolsko in državno tekmovanje v Sloveniji in Srbiji s skoraj 20.000 tekmovalci.

Ključne besede:

porazdeljeni sistemi, tekmovanje Bober, 3-slojna arhitektura, MySQL Cluster, PHP, Yii

Abstract

The thesis describes the implementation of a new platform for the Beaver competition, an international competition in computer thinking and literacy for pre-university students. In past we used Majava competition system. Due to an increase in number participants, Majava implementation and technical layout was found to be unsuitable for use in the upcoming years.

We analyzed existing work in the area of competition platforms, defined quantitative and qualitative requirements, developed a data model, implemented all the necessary functionality, and during production phase provided a fully operational system. The new platform was created using 3-layer distributed architecture, which permits successful distribution of each layer on multiple servers, and thus ensuring high availability and scalability.

In the years 2013/14 we used the system for successful organization of school and national competitions in Slovenia and Serbia with almost 20,000 participants.

Key words:

distributed systems, competition Beaver, three-tier architecture, MySQL Cluster, PHP, Yii

Seznam prevodov in uporabljenih kratic

AJAX (*angl. Asynchronous JavaScript and XML*) – skupina medsebojno povezanih spletnih razvojnih tehnik, uporabljenih za ustvarjanje interaktivnih spletnih aplikacij.

API (*angl. Application programming interface*) – definicija interakcije med posameznimi programskimi komponentami.

CaaS (*angl. Competition as a Service*) – tekmovanje kot storitev

CSS (*angl. Cascading Style Sheets*) – preprosti slogovni jezik, ki skrbi za prezentacijo strani.

DNS (*angl. Domain Name System*) – sistem, ki prevede uporabniku prijazna imena domen v številko IP.

HTML (*angl. HyperText Markup Language*) – označevalni jezik za izdelavo spletnih strani.

JSON (*angl. JavaScript Object Notation*) – odprto-standardni format, ki omogoča zapis objektov v človeško berljivi obliki.

MVC (*angl. model-view-controller*) – model-pogled-nadzor, arhitekturni dizajn

ORM (*angl. Object-relational mapping*) – objektno-relacijska preslikava

PDF (*angl. Portable Document Format*) – standard za izmenjavo elektronskih dokumentov, ki jih lahko prikažemo na vseh platformah.

PHP (*angl. Hypertext Preprocessor*) – odprtokodni programski jezik, ki se uporablja za razvoj dinamičnih spletnih vsebin.

REST (*angl. Representational state transfer*) – preprost način za pošiljanje in prejemanje podatkov med odjemalcem in strežnikom.

SQL (*angl. Structured Query Language*) – strukturirani povpraševalni jezik za delo s podatkovnimi bazami

Poglavje 1

Uvod

1.1 Problem

Diplomsko delo temelji na tekmovanju Bober in težavah, ki so se pojavile med izvajanjem tekmovanja v prejšnjem šolskem letu. Tekmovanje poteka na spletu pod nadzorom mentorjev na posameznih šolah. Največja težava tekmovalne platforme je, da tekmovanje poteka v zelo omejenem časovnem obdobju, to je v enem samem tednu v novembru, kar v praksi pomeni, da v 4 dneh med 8:00 in 12:00 tekmuje večina vseh tekmovalcev. V šolskem letu 2012–2013 je bilo v Sloveniji kar 8.500 tekmovalcev, v naslednjih letih pa se pričakuje, da jih bo čedalje več.

Težava obstoječega tekmovalnega sistema je bila, da ni bilo več mogoče zagotoviti stabilnega delovanja vsem tekmovalcev v Sloveniji, saj sistem ni več omogočal, da bi tekmovanje v enem tednu delovalo brez prekinitev. Problemi so se pojavili predvsem pri strežnikih, ker postavitev sistema nikoli ni bila načrtovana oz. izvedena tako, da bi le-ta lahko bil porazdeljen na več strežnikov, s čimer bi bila omogočena boljša dostopnost do sistema. Obstoječi sistem Majava je razvila skupina finskih študentov v programskem jeziku Ruby, brez uporabe naprednejših ogrodij (*framework*), ki bi omogočili enostavne dodelave in vzdrževanje. V Sloveniji je bil ta sistem v uporabi zadnja tri leta. Skozi leta je raslo število tekmovalcev, hkrati pa se je krčilo

število študentov, ki poznajo osnovo tekmovalnega sistema Majava, kar je privedlo do tega, da je projekt ostal brez vzdrževalcev in brez načina, kako podpreti slovenske potrebe po kapacitetah za izvedbo tekmovanja.

Zaradi povečanja števila tekmovalcev obstoječa administracija v sistemu Majava ne ustreza več potrebam administrativnih uporabnikov, vedno večje so težave s preglednostjo podatkov, pomanjkljivimi funkcionalnostmi glede na potrebe tekmovanja itd. Eno izmed problematičnih področij je tudi varnost, predvsem neobstoj ščitenja vsebin tekmovalnih nalog in način prijave v administracijo, kjer se uporablja enostavna prijava (*BasicAuth*) in kjer so vsa gesla v podatkovni bazi zapisana kot navadno besedilo, brez uporabe zgoščevalnih funkcij (MD5, SHA-1 ...).

1.2 Tekmovanje Bober

Bober je mednarodno tekmovanje v računalniškem razmišljanju in pismenosti za osnovnošolce in srednješolce. Njegov glavni namen je povečati zanimanje učencev za računalništvo in informatiko. S tekmovanjem se otrokom želi pokazati, da računalnik ni le orodje za komuniciranje, brskanje po spletu, urejanje besedil ter poslušanje glasbe in gledanje filmov. Računalnik je namreč neizčrpen vir zanimivih logičnih problemov, zaradi katerih računalničarju ni nikoli dolgčas.

Na tekmovanju otroci spoznajo področje računalništva na zabaven in poučen način. Tekmovalna vprašanja se navezujejo na algoritmično razmišljanje, logično sklepanje, razvoj spretnosti za reševanje problemov, uporabo informacijske tehnologije in njen vpliv v družbi. Otroci ob tem spoznajo, da je razmišljanje in ustvarjalno reševanje problemov zanimivo in zabavno.

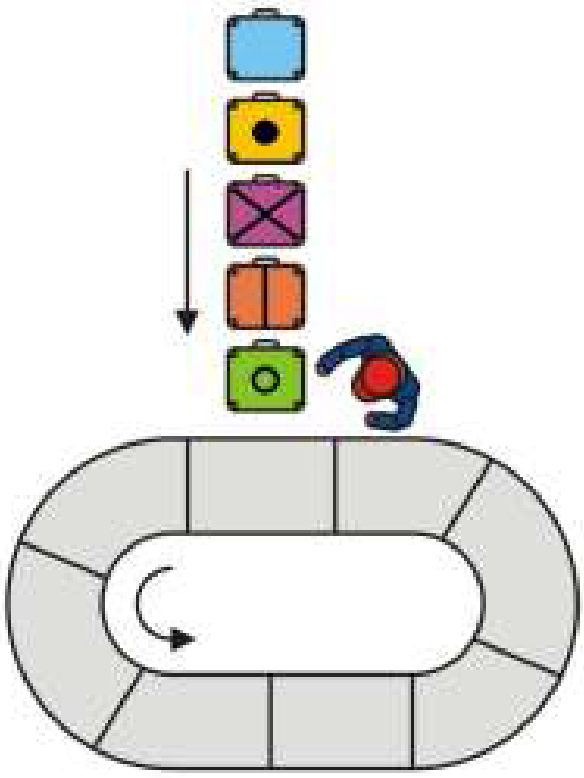
Tekmovanje v Sloveniji poteka v petih kategorijah: Cicibober (prva triada OŠ), Bobrček (druga triada OŠ), Mladi bober (tretja triada OŠ), Bober (1. in 2. letnik SŠ), Stari bober (3. in 4. letnik SŠ). Šolsko tekmovanje je vsako leto organizirano v novembru. Najboljši tekmovalci se uvrstijo na državno tekmovanje v januarju.

Tekmovanje je bilo prvič organizirano v Litvi leta 2004 [52], sedaj pa se izvaja že v 24 državah po vsem svetu [51]. Organizator tekmovanja za Slovenijo je ACM Slovenija [11] v sodelovanju z Univerzami v Ljubljani, Mariboru in na Primorskem.

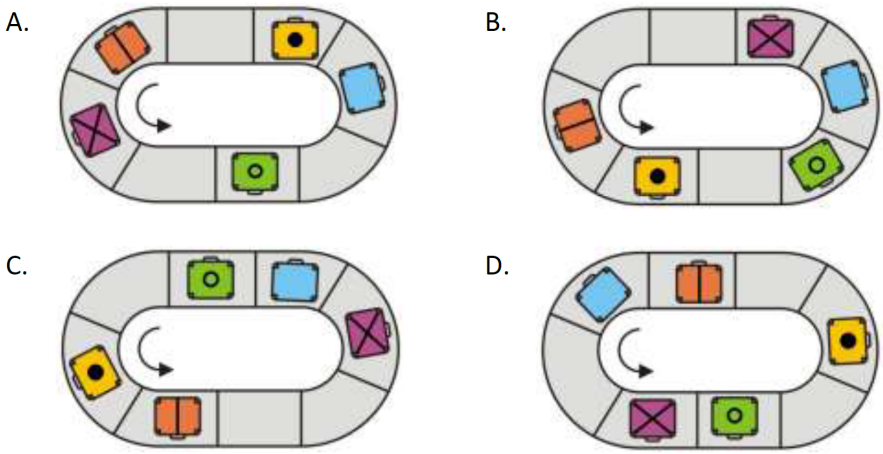
1.3 Primer tekmovalne naloge

Letališče [53]

Na letališču imajo krožni trak za prtljago, ki se vrti, kot je označeno na sliki. Delavec nanj nalaga kovčke, tako da pred vsakim kovčkom izpusti dve prazni mesti (pazi: dve prazni mesti, ne dve mesti) in postavi kovček na tretje prazno mesto.

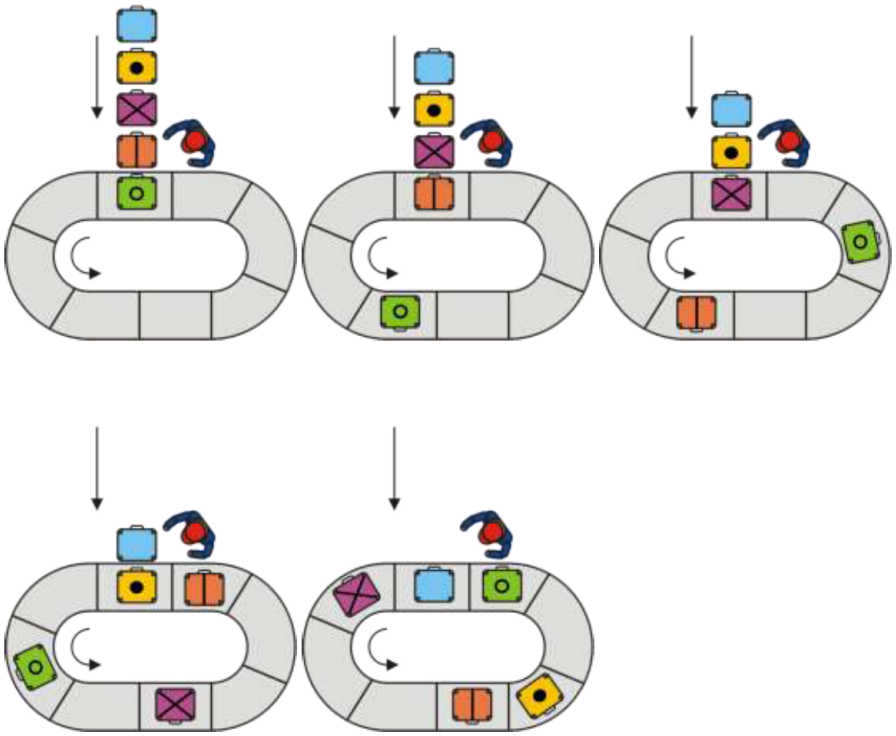


Kako bo videti trak, ko bo delavec nanj naložil vso prtljago s slike?



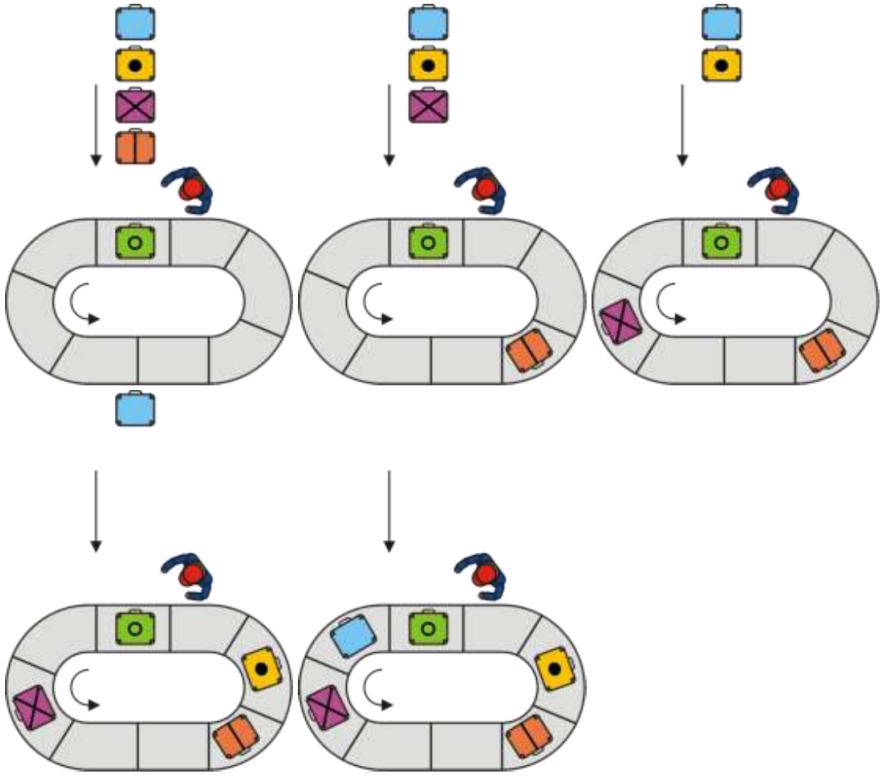
Rešitev: Pravilna rešitev je na sliki B.

Nalogo lahko rešujemo tako, da si predstavljamo, kako se vrti trak in delavec polaga kovčke nanj.



Rezultat je enak rešitvi B, le trak je nekoliko zasukan.

Takšno reševanje ni preprosto – ne na papir, ne na pamet. Veliko enostavneje je, če si predstavljamo, da se trak ne vrti, temveč delavec hodi okrog njega v nasprotni smeri in polaga kovčke na vsako tretje prosto mesto. Rezultat je spet obrnjen nekoliko drugače, a enak.



Računalniško ozadje

Kako računalnik izvaja več – recimo deset – programov hkrati? V resnici tega ne zna. Če ima računalnik procesor z enim samim jedrom (današnji imajo sicer pogosto štiri, vendar to ne reši problema), je to tako, kot bi imel eno samo glavo, ki lahko misli le eno stvar naenkrat. Računalnik tako v resnici ne počne desetih stvari hkrati, temveč le zelo hitro preklaplja med njimi. Za preklapljanje skrbi operacijski sistem (npr. MS Windows, Linux, OS X), ki določa, kateri program bo prišel na vrsto v katerem trenutku – tako kot se delavec v tej nalogi odloča, kateri prostor na traku je na vrsti za naslednji kovček.

V tekmovanju Bober se pojavljalo takšne in podobne naloge, s katerimi se poskuša učence in dijake spodbuditi k logičnem razmišljanju in načrtovanju postopkov. Praktično se tekmovalce pripravi, da se podzavestno začnejo učiti, preden sploh poznajo pojem programiranje ali programirati. Ne gre sicer za sintaktično pravilen zapis programa, ampak kljub temu učenci že pridobivajo potrebna znanja za reševanje zapletenih matematičnih in računalniških problemov, kar bodo potrebovali v skoraj vseh inženirskih poklicih, če se bodo zanje odločili. Prej bodo otroci usvojili ta znanja, večja je verjetnost, da bodo svoje delo v aktivnih letih življenja lažje opravljali in bili morda zaradi tega karierno bolj uspešni.

1.4 Funkcionalne zahteve

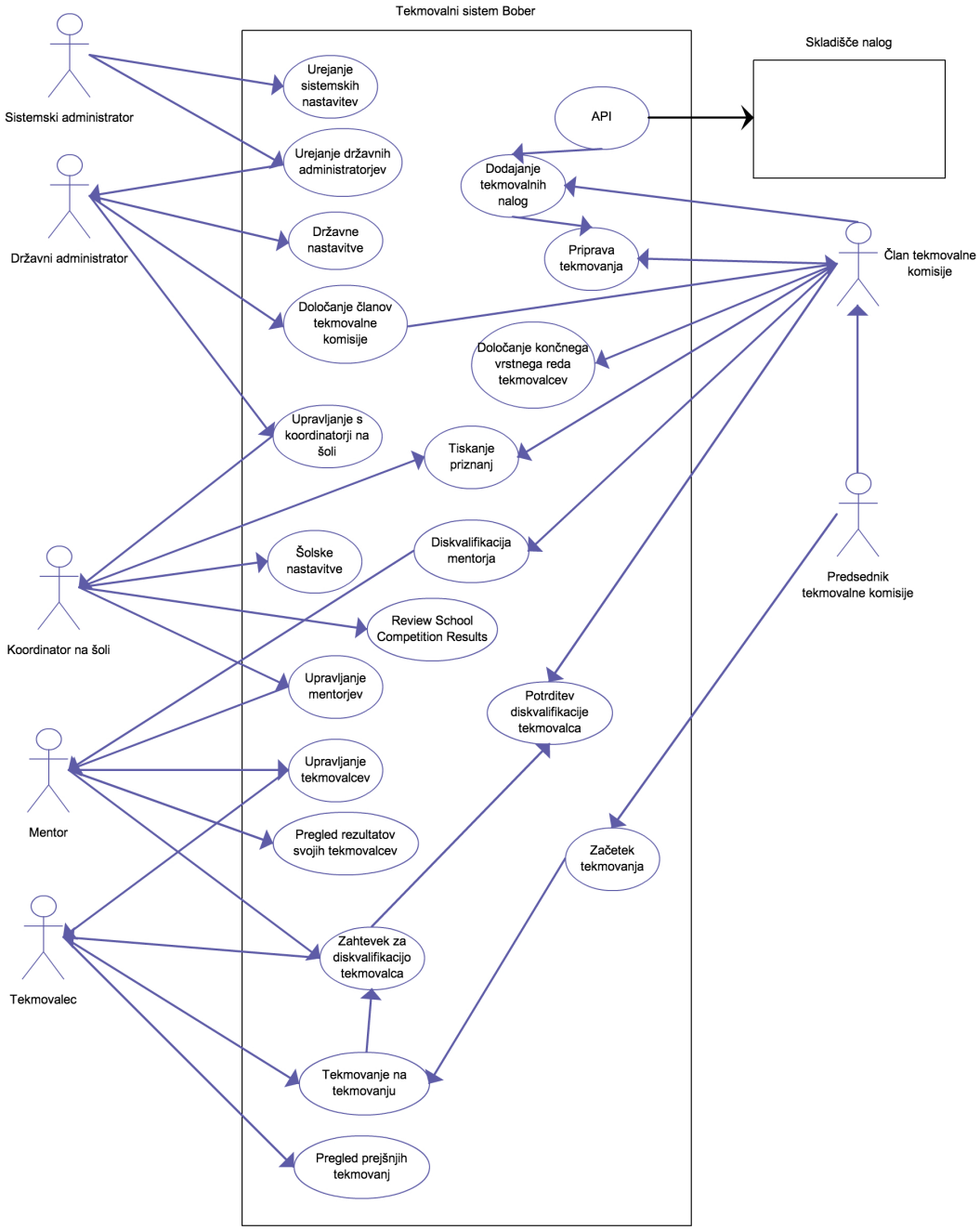
Iz pridobljenega znanja o sistemu Majava in po pogovoru s programskim svetom ACM, ki vodi tekmovanje Bober v Sloveniji, smo oblikovali funkcionalne zahteve za novo platformo. Zahteve smo razdelili na več področij, in sicer na administracijo, mentorstvo in tekmovalce, ter postavili kriterije za kakovost izvedbe projekta.

V sistemu obstaja več vrst uporabnikov, vsak ima svoj nabor funkcionalnosti. Vloge smo razdelili na naslednje: sistemski administrator, državni administrator, koordinator na šoli, mentor na šoli in tekmovalec.

1.4.1 Vloge uporabnikov v sistemu

Sistem mora podpirati dodajanje, urejanje in brisanje uporabnikov. Obstoječim uporabnikom mora omogočati obnovitev pozabljenega gesla. Na sliki 1.1 je izdelana UML (*Unified Modeling Language* oz. poenoteni jezik modeliranja) grafična vizualizacija vlog uporabnikov v sistemu.

Sistemskemu administratorju mora biti omogočen dostop do vseh funkcionalnosti, ki bodo na voljo v celotnem sistemu. Sistemski administrator ima možnost dodajanja novih držav, jezikov, tekmovalnih kategorij, težavnostnih stopenj vprašanj ter upravljanja z vsemi uporabniki v sistemu, njegova glavna



Slika 1.1: Vloge uporabnikov v tekmovalnem sistemu Bober.

naloga pa je dodajanje državnih administratorjev, ki potem administrirajo in urejajo podatke na nivoju države.

Državni administratorji morajo imeti možnost dodajati in spreminjati regije, občine in šole. Za vsako šolo morajo imeti možnost dodajanja mentorjev in določanja koordinatorjev. Na nivoju države imajo možnost pripraviti tekmovanja, pregledovati rezultate in določati meje za udeležbo na naslednjih stopnjah tekmovanja. Državni administratorji morajo imeti tudi možnost ročno prijaviti šole in mentorje na posamezno tekmovanje, odobriti zahteve novih mentorjev za mentorstvo na posameznih šolah in določiti člane tekmovalne komisije.

Člani tekmovalne komisije imajo možnost pripraviti tekmovalne naloge in tekmovanje. Ko imajo pripravljeno tekmovanje, morajo imeti možnost tekmovanje aktivirati in nastaviti datume, po katerih bodo rezultati in priznanja vidni mentorjem. Člani komisije lahko diskvalificirajo posameznega učitelja ali tekmovalca.

Predsednik tekmovalne komisije ima poleg pravic članov tekmovalne komisije še možnost aktivacije tekmovanja, tako da tekmovalci lahko pričnejo s tekmovanjem.

Mentorji se lahko prijavijo za mentoriranje na posamezni šoli. Lahko se prijavijo za sodelovanje na vsakem tekmovanju v državi. Ob prijavi na tekmovanje mentorji dobijo dostopne kode za tekmovalce, ki jim jih v času tekmovanja posredujejo, in na podlagi tega sistem ve, kateri tekmovalec pripada kateri šoli in katerem mentorju. Mentorji imajo poleg tega tudi dostop do rezultatov tekmovanja za svoje tekmovalce in možnost, da rezultate izvozijo za nadaljnjo obdelavo ali tiskanje. Mentor lahko za posameznega tekmovalca zahteva diskvalifikacijo, vendar jo mora nato odobriti eden izmed članov tekmovalne komisije. Mentor je lahko mentor tudi na več različnih šolah.

Koordinatorji so posebna vrsta mentorjev, ki so določeni na posamezni šoli in za katere velja, da imajo dostop do vseh tekmovalcev na šoli, torej tudi do tekmovalcev drugih mentorjev. Posamezni mentorji so lahko koordinatorji na več različnih šolah.

Tekmovalci so posebna vrsta uporabnikov v sistemu, ki nimajo dostopa do administrativnega dela aplikacije, saj lahko uporabljajo zgolj tekmovalni del platforme. Tekmovalcem bi morali omogočiti, da se lahko ločeno registrirajo in svoje tekmovalne podatke povežejo s svojim računom, tako da bodo lahko skozi leta spremljali svoj uspeh.

Sistem mora omogočati tudi javen dostop gostom na osnovno spletno stran, kjer lahko potencialni novi tekmovalci oz. uporabniki, željni novega znanja, preizkusijo tekmovanje, sestavljeno iz preteklih tekmovalnih nalog. Po zaključenem tekmovanju oz. kvizu se mora gostu izpisati rezultat, ki ga je dosegel, ter ponovno prikazati naloge, pri katerih se označijo pravilni odgovori in pripišejo obrazložitve nalog ter njihovo računalniško ozadje.

1.4.2 Opis delovanja tekmovalnega dela platforme

Najbolj pomemben del celotne platforme je tekmovalni del, kjer je pomembno, da je sistem sposoben streči velike količine zahtevkov. Registracija in pričetek tekmovanja morata biti za tekmovalca čim bolj enostavna, zahtevani naj bodo samo res potrebni podatki. Pričetek tekmovanja za posameznega tekmovalca mora biti pogojen s tem, da so vsi potrebni viri za opravljanje celotnega tekmovanja že naloženi v brskalniku, da ne bodo uporabniki s slabšo internetno povezavo časovno prikrajšani. Potrebni podatki za identifikacijo tekmovalca za tekmovanje so: priimek, ime, razred, oddelek, spol in dostopna koda.

Po identifikaciji tekmovalca se morajo naložiti vsi tekmovalni podatki, kar pomeni, da se naložijo vse slike, besedila in skripte, potrebne za tekmovanje. Po končanem nalaganju vseh podatkov mora sistem obvestiti strežnik o zaključku nalaganja in sprožiti tekmovalni čas tekmovalca. Komunikacija med strežnikom in tekmovalčevim brskalnikom mora biti čim manj pogosta in čim krajša. Ob premiku med posameznimi nalogami se mora odgovor na posamezno vprašanje shraniti. Sistem mora prikazovati tudi, koliko je tekmovalcu še ostalo časa, čas pa se mora zaradi čimvečje usklajenosti vsakih 10 minut uskladiti s strežnikom, da je preostali čas tekmovalca čim bolj

usklajen. Strežnik mora skrbeti, da se odgovori izven tekmovalčevega časa tekmovanja ne upoštevajo. V primeru osvežitve tekmovalne spletne strani se morajo ustrezno izbrati že shranjeni odgovori in prikazati preostali čas za tekmovanje. Tekmovalec mora imeti ves čas možnost poljubno navigirati med nalogami. V navigaciji naj bo razvidno, na katera vprašanja je tekmovalec že odgovoril in na katera še ne. Tekmovalec mora imeti možnost izbrisati shranjeni odgovor pri posamezni nalogi. Ob zaključku tekmovanja mora sistem opozoriti tekmovalca o številu neodgovorjenih nalog in o tem, da ob kliku na gumb zaključek ne bo več mogoče spreminjati odgovorov.

Pomembno je, da so vsi viri nalog zaščiteni s tekmovalčevo sejo, torej da se viri naloge (slike, besedilo ...) nalagajo iz URL-naslovov, ki preverjajo, ali ima tekmovalec dostop do tega vprašanja in je tekmovalni čas še aktiven ali ne. Nekdo, ki v danem trenutku ne tekmuje, torej ne sme imeti dostopa do nalog.

1.4.3 Interaktivne naloge

Tekmovanje mora imeti možnost uvoza interaktivnih nalog po standardu Programskega vmesnika za standardne naloge Bober (*Bebras Task Standard API*) [23] oz. po slovenski dopolnjeni različici Navodila za interaktivne naloge Bober [21]. Interaktivne naloge so naloge, ki se jih pripravi izven tekmovalnega sistema Bober in se jih v sistem le uvozi. Sestavljene so s tehnologijami, ki delujejo v spletnih brskalnikih, torej HTML, CSS, JavaScript, Flash itd. Vezni dokument naloge je manifest, to je datoteka, v kateri so zapisani vsi viri naloge, naslov naloge, avtorji in pravilen odgovor oz. funkcija za določanje pravilnega odgovora. Glavna prednost, ki jo prinesejo standardne interaktivne naloge, je možnost, da se naloge uporabljajo v različnih sistemih in da naloge omogočajo dinamično vsebino, kar tekmovalcem omogoča reševanje naloge skozi interaktivno igro.

Interaktivne naloge podpirajo standarden format za komunikacijo med strežnikom in nalogo. Vsaka naloga mora imeti implementirane tri enostavne funkcije, ki jih mora ustrezno klicati tudi tekmovalna platforma, in sicer:

- `task.load(randomSeed, mode)`, ki za posameznega tekmovalca omogoča nastavljanje naključnega števila, od katerega je odvisna vsebina naloge in njeni odgovori, torej mora tekmovalna platforma ob vsaki inicializaciji naloge nastaviti za istega uporabnika enako vhodno naključno številko. Naloga pa mora poskrbeti, da se za enako naključno število prikaže enak vrstni red odgovorov. Ob zaključku te funkcije mora naloga zagotoviti, da je pripravljena za reševanje.

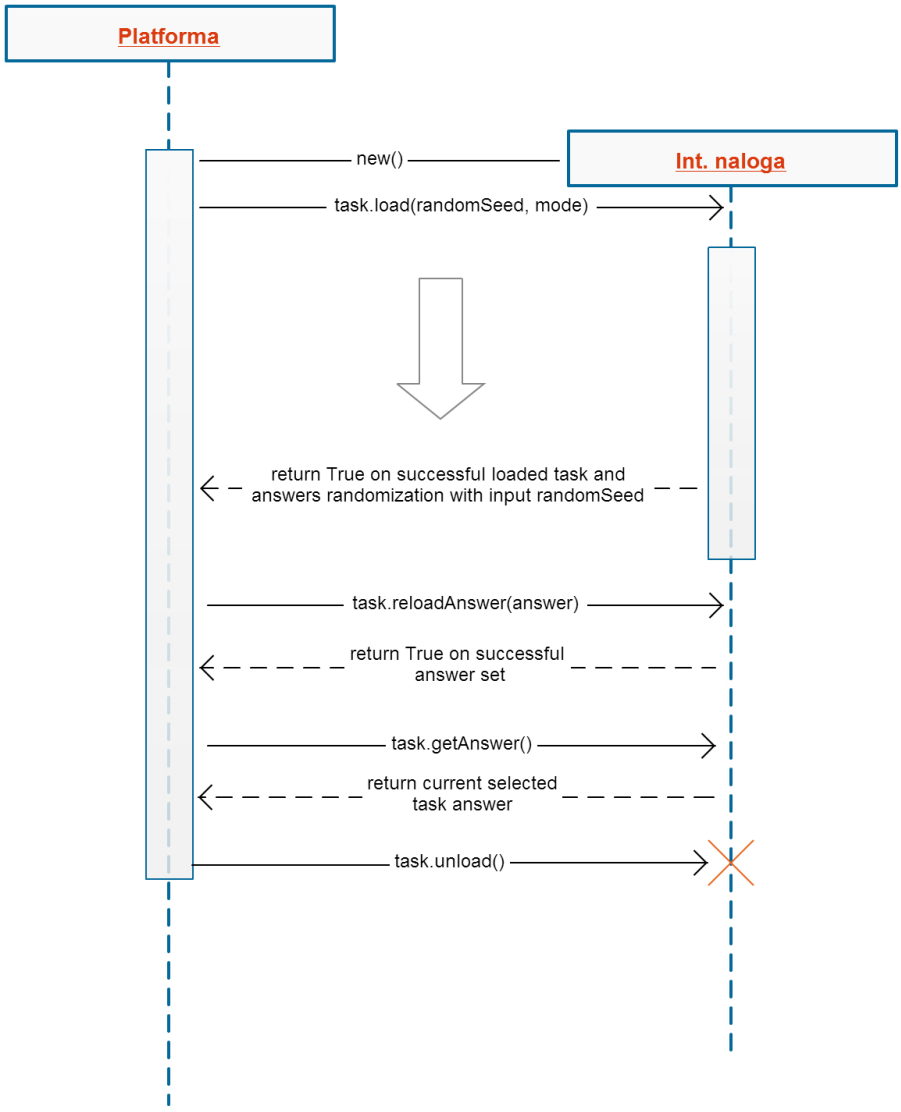
- `task.unload()` je funkcija, ki skrbi za odstranjevanje virov iz brskalnika ob zapiranju naloge, tekmovalna platforma jo mora klicati, ko uporabnik zapušča nalogo.

- `task.getAnswer()` je funkcija interaktivne naloge, ki vrne odgovor, ki ga je tekmovalec izbral v danem trenutku, vrnjeno vrednost mora tekmovalna platforma shraniti in pri ovrednotenju naloge posredovati funkciji interaktivne naloge odgovor za ocenjevanje oz. primerjati odgovor s statično nastavljenim pravilnim odgovorom v manifestni datoteki.

- `task.reloadAnswer(answer)` je funkcija, s pomočjo katere tekmovalna platforma sporoči interaktivni nalogi že prej shranjeni odgovor naloge, tekmovalna platforma mora izvesti ukaz takoj po ukazu `task.load()`, naloga mora ob nastavitvi odgovora s strani tekmovalne platforme izbrati oz. nastaviti ustrezen odgovor znotraj naloge.

Tekmovalna platforma (na sliki 1.2 označena kot „Platforma“) v času reševanja naloge komunicira z interaktivno nalogo (na sliki 1.2 označena kot „Int. naloga“) s pomočjo zgoraj opisanih funkcij interaktivne naloge. Platforma naloži vire naloge (`new()`), nato v kontekstu interaktivne naloge pokliče funkcijo `task.load(randomSeed, mode)` in funkciji v spremenljivki `randomSeed` poda naključno generirano število, ki si ga za posameznega tekmovalca zapomni ter preko spremenljivke `mode` doda parameter k nalogi, preko katerega se nalogi sporoči, če deluje v tekmovalnem načinu ali v vadbenem načinu (vadbeni način lahko prikazuje dodatne pomoči in namige pri reševanju). Naloga na podlagi vhodnih parametrov funkcije

`task.load` ustrezno zgradi vsebino naloge in pomeša odgovore ter platformi odgovori s `True`, če je bila inicializacija naloge uspešna, oz. `False`, če ni bila. Po uspešni inicializaciji naloge platforma pokliče funkcijo naloge



Slika 1.2: Potek komunikacije med platformo in interaktivno nalogo.

`task.reloadAnswer(answer)` in ji preko spremenljivke `answer` poda prej shranjeni odgovor naloge oz. prazno besedilo, kar za nalogo pomeni, da odgovor ni označen. Naloga po uspešnem nastavljanju odgovora odgovori

platformi s `True` za uspešno nastavljanje odgovora oz. `False` za neuspešno. Platforma isti klic funkcije `task.reloadAnswer(answer)` uporablja za dva različna namena, in sicer za ponovno nalaganje že predčasno shranjenega odgovora ali pa za brisanje odgovora na nalogo (`answer` je enako prazno besedilo). Ko uporabnik preklopi med nalogami oz. ko platoforma želi shraniti odgovor, platforma pokliče funkcijo naloge `task.getAnswer()`, naloga kot rezultat funkcije vrne trenutno izbrani odgovor oz. vpisano rešitev naloge. Rezultat, ki ga vrne funkcija `task.getAnswer()`, mora biti v taki obliki, da lahko ista naloga sprejme v spremenljivki `answer` pri klicu `task.reloadAnswer(answer)`. Po končani komunikaciji med platformo in interkativno nalogo oz. ko uporabnik zapusti nalogo, je platforma dolžna poklicati funkcijo `task.unload()`, ki nalogi sporoči, naj sprosti vse uporabljene vire v pomnilniku in zaključi delovanje.

1.4.4 Točkovanje tekmovanja in priznanja

Tekmovalni sistem mora omogočati izgradnjo tekmovanja. Tekmovanje je sestavljeno iz nalog, vsaka tekmovalna naloga mora imeti določeno, v katero tekmovalno kategorijo sodi (Bobrček, Mladi Bober, Bober, Stari Bober) in kakšna je njena težavnostna stopnja (lahka, srednja, težka). Sistem mora omogočati, da se lahko poljubno dodajajo še nove tekmovalne kategorije in težavnostne stopnje. Glede na težavnostno stopnjo naloge v posamezni tekmovalni kategoriji se določi, kaj posamezna težavnostna stopnja pomeni pri vrednotenju nalog. Vsaka težavnostna stopnja mora za pravilen odgovor na vprašanje imeti določeno število točk, prav tako mora imeti določeno število negativnih točk v primeru napačnega odgovora. Če tekmovalec ne odgovori na vprašanje, ne dobi točk. Sistem mora poskrbeti, da končen seštevek točk nikoli ni negativen, tudi če tekmovalec vse odgovori narobe. Začetno število točk tekmovalca je enako absolutni vsoti vseh možnih negativnih točk, ki bi jih tekmovalec dobil, če bi na vse naloge odgovoril napačno.

Na podlagi točkovanja se pripravijo rezultati po posameznih kategorijah tekmovanja. Kriterij za podelitev bronastih priznanj je točno določen s pra-

vilnikom tekmovanja [7], tako da priznanje dobi tekmovalec, ki se uvrsti v prvo petino najboljših tekmovalcev po skupnem številu doseženih točk v posamezni tekmovalni kategoriji na državni ravni ali ki se uvrsti med najboljšo tretjino tekmovalcev na šoli v posamezni tekmovalni kategoriji po skupnem številu doseženih točk in hkrati doseže vsaj polovico vseh možnih točk na tekmovanju. Vse meje števila točk pri določanju, ali tekmovalec pade v delež prejemnikov priznanja ali ne, so zaokrožene navzdol. Sistem mora omogočati preračun priznanj (ob preračunu se morajo obstoječa priznanja tekmovalcev pobrisati), avtomatsko določitev priznanj ter možnost ročnega določanja, kdo dobi priznanje. Sistem mora omogočati tudi pregled, seštevek priznanj po mentorjih, šolah itd.

Določanje napredovanja na naslednjo stopnjo tekmovanja mora omogočati, da se na podlagi razpoložljivih tekmovalnih mest na naslednji stopnji tekmovanja določi, od katere točkovne vrednosti naprej se bodo tekmovalci udeležili tekmovanja. Vsak ponovljeni preračun mora trenutno določene tekmovalce za napredovanje pobrisati in jih na novo določiti. Sistem mora omogočati tudi ročno določitev tekmovalcev, ki se bodo udeležili tekmovanja na naslednji stopnji tekmovanja.

Rezultatov tekmovanja, priznanj in podatka, kateri tekmovalci napredujejo na naslednjo stopnjo tekmovanja, se ne sme prikazati mentorjem in koordinatorjem, kljub temu da so izračuni že narejeni. Za vsako izmed naštetih stvari mora v administraciji tekmovanja obstajati časovni žig, ki določa datum, od katerega dalje so podatki vidni.

Sistem mora omogočati, da si mentorji/koordinatorji lahko preko tekmovalne platforme prenesejo priznanja in si jih natisnejo sami na šoli. Datoteke s priznanji pripravi tretji sistem, ji naloži na tekmovalno platformo in dâ na voljo PDF datoteke. Tekmovalna platforma pa mora poskrbeti, da datoteke pridejo do ustrezno pooblaščenih mentorjev in koordinatorjev.

1.4.5 Kvalitativne in kvantitativne zahteve delovanja

Sistem mora imeti naslednje lastnosti:

- zadoščeno mora biti vsem funkcionalnim zahtevam,
- podpirati mora možnost, da naenkrat tekmuje vsaj 10.000 tekmovalcev,
- zmogljivost 100.000 tekmovalcev, porazdeljenih čez 5 dni tekmovanja,
- maksimalen čas vzpostavitve sistema v primeru popolnega izpada ne sme presegati 20 minut. Popoln izpad pomeni, da ne ostane prižgan noben strežnik v porazdeljenem sistemu, kar pomeni, da se mora zadostna količina strežnikov ponovno zagnati in pred nadaljevanjem dela med seboj sinhronizirati podatke.

Poglavje 2

Obstoječe delo

V svetu obstaja veliko sistemov za tekmovanje, na področju tekmovanja Bober je znan predvsem Majava (sistem na Finskem), sistem v Franciji, sistem v Latviji in komercialni projekt v Nemčiji. O sistemu v Nemčiji in o njegovi arhitekturi zaradi zaprtosti oz. nedostopnosti kode in literature ne vemo praktično ničesar, vendar na leto z njim opravi tekmovanje preko 150 tisoč tekmovalcev v Nemčiji, Avstriji, Kanadi in na Nizozemskem.

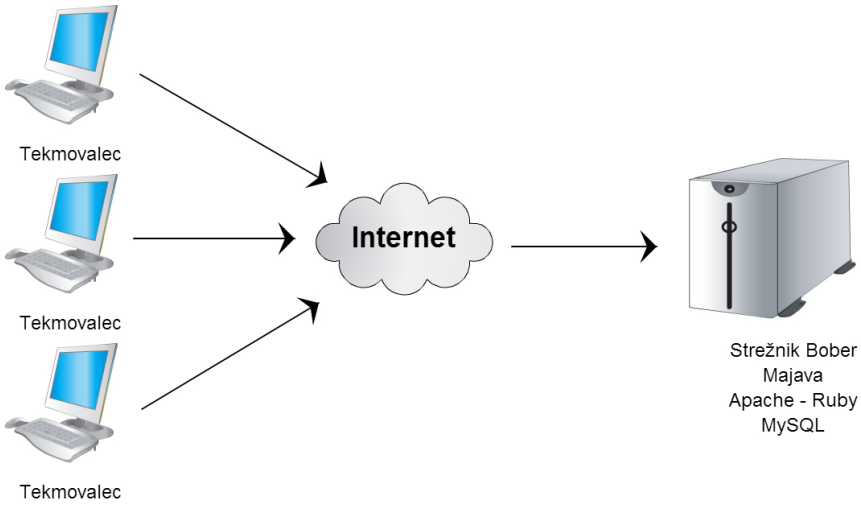
Na področju učnih sistemov, ki niso neposredno usmerjeni v reševanje problematike tekmovanja Bober, obstaja kar nekaj dobro zasnovanih platform.

2.1 Majava

Obstoječi sistem Majava, ki se je v Sloveniji uporabljal od začetka tekmovanja Bober do sedaj, je naredila skupina finskih študentov in sloni na programskem jeziku Ruby, brez uporabe naprednejših ogrodij, ki bi omogočila enostavne dodelave in vzdrževanje.

Osnovna in največja težava sistema Majava je njegova arhitekturna zasnova, saj razvijalci niso razmišljali o razširljivosti sistema. Arhitekture niso razdelili na več slojev, ampak so enostavno vse sloje prepletali med seboj, kar je privedlo do omejenosti na uporabo samo ene vrste podatkovnega skladišča,

v tem primeru MySQL, in neoptimiziranega dostopa odjemalcev do strežnika. Majava je bil v osnovi zastavljen tako, da so vse plasti nameščene na en sam strežnik (glej sliko 2.1), kar pa predstavlja probleme pri povečljivosti zaradi večjih potreb pri večjem številu tekmovalcev. Ker smo poznali trenutne težave sistema Majava, je bil naš načrt – pripraviti funkcionalno enakovreden in arhitekturno boljši sistem – trdno začrtan.



Slika 2.1: Komunikacija med tekmovalci in sistemom Majava (samo en strežnik).

2.2 Sistem v Franciji

Francozi so pripravili svoj sistem, a ko smo ga podrobno pregledali, smo ugotovili, da je arhitekturno zgrajen podobno kot Majava, vmesna plast je prav tako konkretno vpletena v podatkovno sloj, v kodo so neposredno zapisane SQL-poizvedbe. Dobra stran sistema je optimizacija tekmovalnega dela, kjer je optimizirano nalaganje vsebin in poizvedb na strežnik. Kljub dobro narejenem tekmovalnem delu je problematičen njihov administrativni del, ki ne omogoča nobenih naprednih funkcionalnosti, ki bi jih pri našem sistemu želeli imeti. Francoski sistem arhitekturno ni zasnovan kot porazdeljen sis-

tem. Trenutno podpira samo enostrežniško podatkovno bazo. Imajo pa več spletnih strežnikov, da med njih porazdelijo zahtevke uporabnikov. Seje v sistemu so implementirali s pomočjo strežnika za predpomnjenje *memcache* in na ta način omogočili enotne seje, porazdeljene med vse spletne strežnike. Sistem nima postavljene nobene replikacije, razen na popolnoma strojnem nivoju, kjer uporabljajo zrcaljenje diskov RAID 1. Letos so dodali še dodaten varnostni mehanizem za zapisovanje odgovorov tekmovalcev v tekstovne datoteke na drug strežnik, da v primeru izpada SQL-strežnika ne izgubijo podatkov.

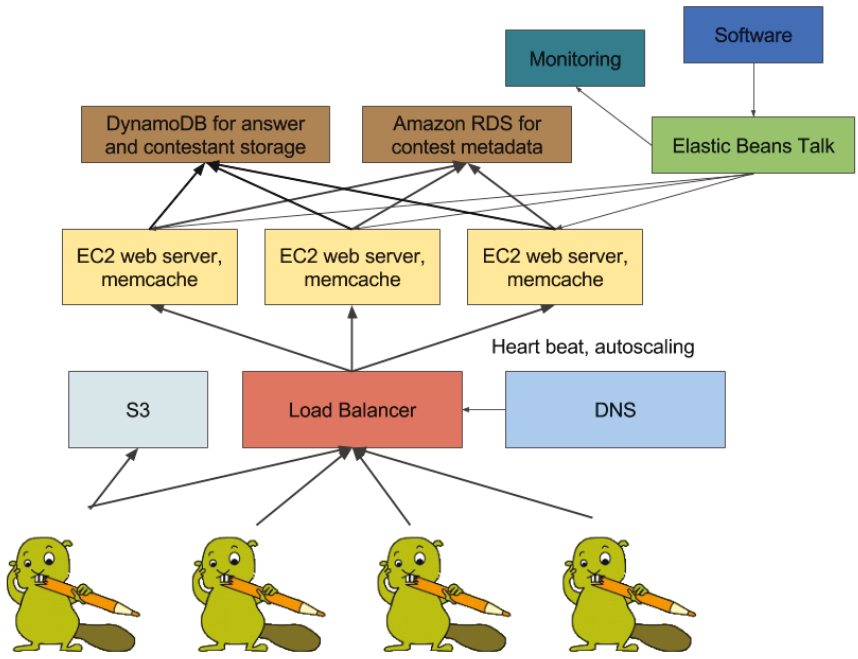
Zanimiv pa je tudi njihov pristop v primeru, ko noben strežnik ni več dostopen oz. tekmovalec ostane brez interneta, saj se tekmovalcu na koncu pojavi kodirano besedilo, ki ga lahko pošlje po elektronski pošti, tako se odgovore v sistem uvozi naknadno. Ker so sistem želeli pripraviti, tako da bi lahko podprl kar se da največ z enostavno infrastrukturo, so omejili komunikacijo s strežnikom, tako da se podatki oz. odgovori na vprašanja pošljejo samo ob koncu tekmovanja, kar zna biti problematično v primeru sesutja spletnega brskalnika [22].

2.3 Sistem v Latviji

Med drugim so tudi Latvijci pripravili svoj sistem, ki je nekoliko bolj razdelan in arhitekturno razslojen, tako da omogoča namestitev na Amazonov oblak.

Na sliki 2.2 je prikazana shema delovanja sistema v Latviji, sistem je postavljen na EC2-virtualnih strežnikih na Amazonovem oblaku. Na EC2-virtualnih strežnikih imajo nameščene spletne strežnike in strežnike za predpomnjenje (*memcache*). Uporabniške zahtevke so porazdelili med virtualne strežnike z uporabo kombinacije DNS-strežnikov in izenačevalnika obremenitve (*load balancer*). Trajnostno shranjevanje podatkov so uredili s pomočjo S3-sistema za shranjevanje podatkov. DynamoDB-podatkovno bazo so uporabili za hitro shranjevanje in branje tekmovalnih podatkov (podatki o tekmovalcih, odgovori na naloge). Za manj pogosto pisanje in branje pa so

uporabili podatkovno bazo Amazon RDS, v katero zapisujejo predvsem metapodatke tekmovanja in nalog. Vklapljanje dodatnih strežnikov v primeru visoke obremenitve so rešili z Amazonovo storitvijo Elastic Beans Talk, ki omogoča preverjanje stanja virtualnih strežnikov in avtomatski zagon novih. Vse storitve, ki jih imajo, spremljajo (*monitoring*), da lahko hitreje zaznajo izpade in težave. S to arhitekturno postavitvijo, vsaj tako trdijo, lahko dosežejo tudi do 100.000 tekmovalcev na uro [5] [4].



Slika 2.2: Shema delovanja Latvijskega sistema Bober na Amazonovi platformi [3].

2.4 Ostali sistemi

Obstaja več odportokodnih rešitev za izobraževalne namene (LMS-sistemi), ki niso specifično usmerjene v neko vrsto tekmovanja. Mednje sodijo spletne učilnice Moodle [32], ILIAS [24], DotLRN [29], Openelms [12]. Vsi našteti sistemi uporabljajo večslojno arhitekturo, ki jim omogoča prilagodljivost na

različnih slojih in načeloma, z manjšimi spremembami v konfiguraciji, tudi razširljivost na več strežnikov. Najpomembnejša arhitekturna zasnova vseh teh obstoječih priznanih sistemov je, da omogočajo ločitev na minimalno tri sloje sistema: ločen način prikaza (teme), vmesna plast (programska logika) in podatkovna plast (kjer vsi sistemi omogočajo priklop na različne podatkovne baze: MySQL, postgreSQL, SQLite ...).

Italija uspešno izvaja tekmovanje Bober s sistemom Moodle z vgrajenim orodjem Kviz [8].

Ti sistemi so splošni in omogočajo veliko stvari, a tekmovanje Bober je nekaj posebnega, kar se tiče zahtev in specifikacij tekmovanja, zato potrebuje nek nov sistem, ki bo omogočil dobro administracijo in hkrati zadostil potrebe po kapaciteti zaradi števila hkratnih tekmovalcev na sistemu, zato smo se odločili za pripravo novega sistema, ki bo upošteval vse težave in problematiko tekmovanja.

Poglavje 3

Uporabljene tehnologije

3.1 Porazdeljeni sistemi in doseganje visoke razpoložljivosti

Porazdeljeni sistemi so sistemi, ki so zasnovani tako, da celoten sistem ni prisoten samo na enem fizičnem strežniku, ampak so posamezni deli na različnih strežnikih oz. so posamezni deli podvojeni na različnih strežnikih. Cilj porazdeljenega sistema je večja moč procesiranja in hkrati večja odpornost na odpoved posameznih delov celotnega sistema. Porazdeljeni sistemi skrbijo, da je želena vsebina dostavljena do uporabnika, a uporabnik o porazdeljenosti sistema ne ve praktično ničesar.

Arhitekturno razdeljevanje na več slojev omogoča lažjo porazdelitev sistema. Več slojev omogoča lažjo kontrolo delov sistema in nadomeščanje delov sistema, ki ne delujejo, s preostalimi delujočimi deli porazdeljenega sistema.

Najbolj znana razdelitev porazdeljenega sistema je 3-slojna arhitektura [13]. 3-slojna arhitektura je sestavljena iz:

- **predstavitvenega sloja**: pogledi in kontrole pri odjemalcu,
- **aplikacijskega sloja**: vsa logika delovanja aplikacije, obdelave podatkov za odjemalca, skrb za komunikacijo med odjemalcem in podatkovnim slojem, in

- **podatkovnega sloja**: skrb za shranjevanje podatkov, transakcije, replikacije, fragmentacije itd.

Vsak sloj v arhitekturi se lahko razdeli še na več modulov, ampak običajno so to že skupki programov, ki poskrbijo, da nek sloj deluje tako, kot mora. Za doseganje večje procesne moči in redundance v primeru izpada se na vsakem sloju probleme rešuje na podoben način. Aplikacijski sloj, ki na primer potrebuje več procesorske moči za obdelavo in pripravo podatkov odjemalcu, lahko enostavno požene več primerkov istega aplikacijskega strežnika na različni strojni opremi. S tem se doseže, da lahko obdelamo večje število zahtevkov odjemalcev. Na podatkovnem sloju pri porazdeljenem sistemu skrbimo predvsem za večjo zanesljivost zapisanih podatkov in hitrejši bralno-pisalni dostop [10]. Več o porazdeljenem podatkovnem sloju je opisano v poglavju 3.2.

Porazdeljeni sistemi in doseganje visoke razpoložljivosti je v današnjem času vedno bolj pomembna zadeva, ker ima večina današnjih poslovno zanimivih storitev veliko uporabnikov, veliko uporabnikov pomeni veliko zahtevkov, veliko zahtevkov pa pomeni preseganje zmogljivosti posameznega strežnika. Ko storitev za svoje delovanje potrebuje več strežnikov, se poleg porazdeljevanja zahtevkov pojavi tudi težava, kako usklajevati podatke med seboj, ko je en podatek na enem koncu sveta in drug na drugem [30].

S storitvami, ki uporabljajo tehnologije porazdeljenih sistemov, so uspela znana svetovna podjetja, kot so Google, Amazon, SAP, Yahoo, Microsoft Azure. Google je sprva svojo infrastrukturo uporabljal samo v lastne namene, nato pa je leta 2008 omogočil uporabo vsem v obliki storitve Google App Engine, ki kot storitev glede povečljivosti velja za eno izmed cenejših platform (*PaaS, Platform as Service*) [48], saj omogoča nizke začetne stroške (začetna količina zastonj [18]), in kljub temu zagotavlja, da storitve, ki so razvite na tej platformi, lahko dosežejo večje zmogljivosti brez dodatnega programiranja in skrbi razvijalcev za infrastrukturo. Slabost porazdeljenega sistema Google App Engine je, da zahteva uporabo Googlovih tehnologij, knjižnic, predvsem pa podatkovnih baz, kar pomeni, da je treba razvijati programsko kodo, ki

bo lahko uporabljena samo na njihovi platformi in nikjer drugje.

Za razliko od Googlove specifične platforme kot storitve je Amazon ponudil veliko bolj prilagodljiv sistem, in sicer infrastrukturo kot storitev (*IaaS, Infrastructure as Service*). To je sistem virtualnih strežnikov, kjer je mogoče vsak strežnik prilagoditi lastnim potrebam in potrebam množice različnih podatkovnih baz. Amazon omogoča lažji prehod med lastnimi strežniki in porazdeljenimi strežniki, vendar za višjo ceno storitve, kakor tudi veliko večjo kompleksnost pri postavljanju storitve, veliko več ročnih nastavitev, konfiguracije virtualnih strežnikov [46] itd.

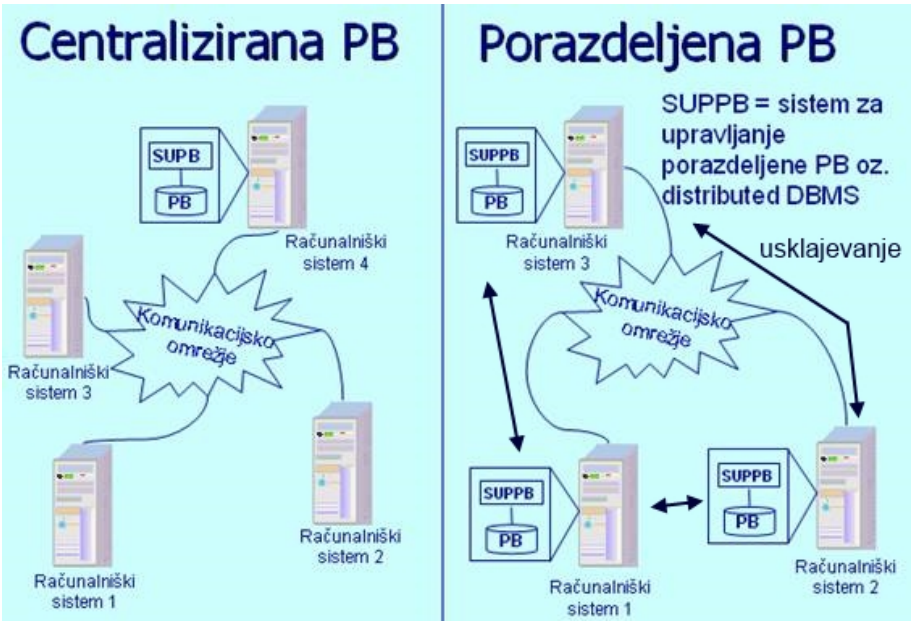
Poleg komercialnih porazdeljenih sistemov so se v odprtokodnem združenju pojavili tudi nekateri sistemi, ki jih velja omeniti – Apache Storm, QADPZ ... Apache Storm (*distributed and fault-tolerant realtime computation* – porazdeljeno realnočasovno računanje, odporno na napake) je sistem, ki omogoča porazdeljeno realnočasovno računanje. Omogoča enostavno in zanesljivo obdelavo tokovnih podatkov. V Storm-sistemu se lahko uporablja kateri koli programski jezik, sistem pa se večinoma uporablja za sprotne analitike, strojno učenje, porazdeljene RPC-sisteme [2]... QADPZ (*Quite Advanced Distributed Parallel Zystem*) je sistem za porazdeljeno računanje. Sistem omogoča nadzor in uporabo računske zmogljivosti več računalnikov v mreži. Uporabniki lahko v sistem pošiljajo naloge, ki so nato izvedene na različnih računalnikih v sistemu. Naloge so lahko v obliki dinamičnih knjižnic, izvedljivega programa ali katerega koli programa, ki ga je mogoče interpretirati (Java, Perl ...). Podprte so platforme Linux, Unix, Windows in MacOS X [33].

Pri vseh navedenih odprtokodnih rešitvah težave predstavljajo pomanjkanje dokumentacije, slaba ali neobstoječa podpora, potreba po sredstvih za nakup velike količine strojne opreme in dejstvo, da je vse treba samostojno preučiti. A tudi, ko je sistem že postavljen, ga je treba ves čas vzdrževati, predvsem strojno opremo, za katero sicer v primeru najema ni treba skrbeti. Zaradi vseh teh slabosti je večina malih podjetij [9] pripravljena najeti zunanje izvajalce, ki imajo svoje lastne strokovnjake in infrastrukturo, da opravijo

delo namesto njih, s tem pa se sami lahko bolj posvetijo logiki aplikacije in uporabniški izkušnji, namesto da se obremenjujejo s tem, kako zadeve delujejo na nivoju operacijskega sistema in strojne opreme.

Pri vsakem postavljenem porazdeljenem sistemu pa želimo meriti njegovo stabilnost, razpoložljivost in zmogljivost. Na tem področju je bilo razvitih več orodij, in sicer ApacheBench, Apache JMeter, Curl-Loader, Httperf, OpenSTA. Vsa ta orodja omogočajo testiranje, koliko zahtevkov lahko streže posamezna storitev v nekem časovnem obdobju, koliko hkratnih zahtevkov lahko streže in še veliko več. S testiranjem zmogljivosti strežnikov se načeloma hkrati testira tudi obremenjenost sistema podatkovnih baz, predpomnjenja itd.

3.2 Porazdeljene podatkovne baze in rešitve



Slika 3.1: Shematski prikaz centralizirane in porazdeljene podatkovne baze [28].

Porazdeljene podatkovne baze so podatkovne baze, pri katerih se podatki ne nahajajo samo na enem strežniku (na sliki 3.1 levo), ampak so podatki shranjeni na več strežnikih (na sliki 3.1 desno), ki se lahko nahajajo tudi na različnih fizičnih lokacijah, oz. so podatki razpršeni čez mrežo povezanih računalnikov. Za razliko od tradicionalnih centraliziranih podatkovnih baz, kjer se vse nahaja na enem fizičnem računalniku (SUPPB in PB) in so vse programske komponente podatkovne baze implementirane z mislijo na lokalno izvajanje (podatki samo na eni lokaciji, podatki dostopni neposredno brez dodatne mrežne komunikacije, brez dodatne logike za porazdelitev zahtevkov), so pri porazdeljenih podatkovnih bazah podatki oz. moduli (samostojni kosi programske opreme, ki so med seboj neodvisni in samostojni) celotnega sistema podatkovne baze porazdeljeni na več računalnikov (SUPPB in PB se nahaja na več različnih sistemih). Posamezni moduli podatkovne baze se med seboj povezujejo le preko komunikacijskega omrežja, brez neposredne fizične povezave. Posamezen modul skrbi za svoje funkcionalnosti in lasten sloj abstrakcije, recimo dostop do podatkov v shrambi, izvajanje obdelav nad podatki, izgradnja indeksov, skrb za konsistenco.

3.3 HTML in CSS

HTML (*Hyper Text Markup Language*) je označevalni jezik za izdelavo spletnih strani. Predstavlja osnovo spletnega dokumenta. S pomočjo HTML poleg prikaza dokumenta v brskalniku hkrati določimo tudi strukturo in semantični pomen delov dokumenta. HTML je skratka prikazni jezik, v katerem se zapiše vsa vsebina, ki jo lahko nato uporabnikov brskalnik interpretira in grafično prikaže.

CSS (*Cascading Style Sheets*) oz. po slovensko kaskadne stilske podloge je opisni jezik, ki skrbi za prikaz spletnih strani. Z njim definiramo stil HTML elementov v smislu pravil, kako naj se elementi prikažejo na strani. V CSS lahko določimo barve, velikosti, odmike, poravnave, obrobe, pozicije in vrsto drugih atributov, prav tako pa lahko nadziramo aktivnosti, ki jih uporabnik

nad elementi na strani izvaja (npr. prekritje povezave z miško). CSS je bil razvit z namenom konsistentnega načina podajanja informacij o videzu spletnih dokumentov. CSS se uporablja predvsem za ločitev strukture strani od njene predstavitve/oblike.

3.4 JavaScript

JavaScript je objektni skriptni programski jezik, ki ga je razvil Netscape, da bi spletnim programerjem pomagal pri ustvarjanju interaktivnih spletnih strani. Jezik je bil razvit popolnoma neodvisno od Jave, vendar si z njo deli številne lastnost in strukture. JavaScript lahko sodeluje s HTML-kodo in s tem poživi stran z dinamičnim izvajanjem. JavaScript podpirajo praktično vsi spletni brskalniki. Sintaksa jezika JavaScript ohlapno sledi programskemu jeziku C. JavaScript je pomemben na vsaki spletni strani, na kateri želimo imeti dinamično naloženo in prilagojeno vsebino. Programsko kodo spletni brskalniki interpretirajo in poženejo v peskovniku v kontekstu spletne strani, s katere je bila naložena koda. JavaScript pogosto zamešajo s programskim jezikom Java, vendar sta si jezika med seboj zelo različna. Bistvena razlika med njima je v izvajanju, saj Java za izvajanje v brskalniku potrebuje dodatno programsko opremo, ki programsko kodo pred izvajanjem prevede v strojni jezik, JavaScript pa se lahko začne izvajati in interpretirati takoj, ko brskalnik prenese izvorno programsko kodo. V našem projektu je JavaScript zelo pomembna komponenta, ker omogoča preložitev izračuna in dela na uporabnikove brskalnike in s tem minimizira obremenitev strežnika z izpisom HTML-vsebine, prav tako se zmanjša velikost zahtevkov v obe smeri, proti strežniku in proti uporabniku.

3.5 PHP

Glavni programski jezik, v katerem so bile implementirane vse funkcionalnosti sistema aplikacijske plasti našega projekta, je odprtokodni jezik PHP.

PHP je kratica za programski jezik po imenu *Hypertext Preprocessor*, izvorno ime pa je bilo dejansko *Personal Home Page Tools*, po slovensko orodje za osebno spletno stran. Programski jezik PHP se uporablja predvsem za razvoj dinamičnih spletnih vsebin. Dinamične spletne vsebine so vsebine, ki so na voljo na spletu in imajo v ozadju programsko kodo, ki spreminja vsebino spletne strani s podatki, ki so bodisi zapisani v podatkovni bazi bodisi v kakšnem drugem podatkovnem viru. Po funkcionalnosti je podoben Microsoftovim sistemom ASP, VBScript in Sub Microsystemovim sistemom JSP in Java ter sistemom CGI in Perl. Podoben je običajnim strukturiranim programskim jezikom, najbolj pa je podoben C-ju in Perl-u in izkušenim programerjem dejansko omogoča, da lahko začnejo razvijati brez dolgega učenja.

PHP je bil napisan kot skupina CGI-programov v programskem jeziku C. Napisal ga je dansko-kanadski programer Rasmus Lerdorf leta 1994, zgolj za potrebe izgradnje enostavnih domačih spletnih strani. Trenutno so v uporabi tri večje različice, in sicer 5.3, 5.4 in 5.5, v pripravi pa je že različica 5.6 [41].

PHP-interpreter primarno teče na spletnem strežniku, kjer za vhod jemlje izvorno kodo PHP in ustvari spletno stran za izhod. Del PHP-ja sta tudi možnost zaganjanja skript v ukaznem načinu CLI (*command line interface*) in kreiranja grafičnih aplikacij. V zadnjem času je bilo na strani strežnikov pripravljenih veliko optimizacij in različnih načinov izvajanja: vgrajen PHP-interpreter modul v apache2, CGI-modul, FPM-modul (uporabljen tudi v našem produkcijskem okolju) in naravnost v PHP vgrajen spletni strežnik kot samostojni strežnik [42]. V preteklosti je bila zelo znana optimizacija PHP, ki jo je naredila družba Facebook, to je HipHop-prevajalnik (odprtokoden), ki dejansko PHP-kodo prevede v C in nato neposredno v binarno datoteko. Facebook trdi, da prevedene zadeve delujejo tudi do šestkrat hitreje in rabijo veliko manj virov [14].

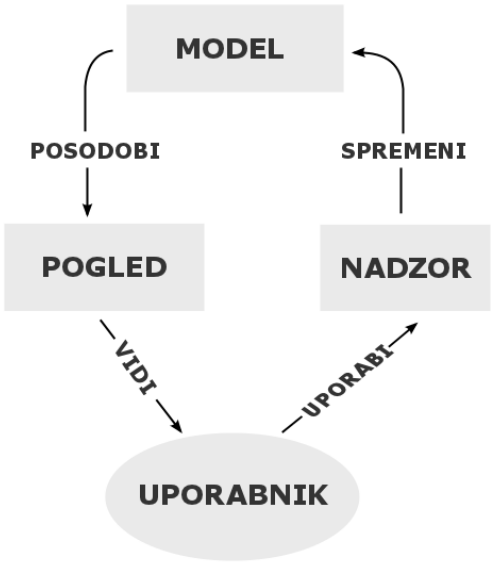
Programski jezik je enostaven jezik in omogoča sproščeno, netipizirano programiranje, kar omogoča implementacijo dinamičnih algoritmov, ki se za različne tipe vhodnih podatkov različno obnaša, kar pri večjih projektih predstavlja nevarnost, saj te lastnosti večkrat kot koristi prinesejo težave. Pro-

gramski jezik je razširjen in priljubljen predvsem pri spletnih projektih, ker omogoča hitro programiranje in veliko prožnosti. Jezik omogoča tako funkcijsko (programska koda je strukturirana s funkcijami, ki imajo določen vhod in izhod) kot tudi objektno orientirano (programska koda je strukturirana z objekti, vsak objekt ima lahko določene vrednosti in funkcije, ki se izvajajo nad tem objektom) programiranje. Odločitev o načinu programiranja je povsem prepuščena razvijalcem.

V PHP je nastalo veliko zelo uspešnih in znanih projektov: Joomla (sistem za upravljanje vsebin) [50], Wordpress (spletni program za bloge, za objavo člankov) [54], Moodle (spletne učilnice, po vsem svetu v uporabi predvsem za vodenje učnega procesa in vsebin) [32].

3.6 Ogrodje Yii - MVC

Yii je ogrodje za razvoj spletnih aplikacij v programskem jeziku PHP. Ime Yii izvira iz akronima ,,Yes It Is!“ oz. slovensko ,,Ja to je!“ kot odgovora na vprašanje, ali je ogrodje Yii hitra, varna, profesionalna in pravilna izbira za naslednji projekt. Začetki ogrodja segajo že v leto 2008 in ogrodje se ves čas dopolnjuje in nadgrajuje. Najbolj pomembna funkcionalnost ogrodja je njegov MVC-arhitekturni dizajn [19]. MVC pomeni *model* – *view* (pogled) – *controller* (nadzor). MVC-razdelitev v praksi pomeni, da obstaja razdelitev na tri logično ločene plasti v programju (glej sliko 3.2), in sicer model, nadzor in pogled. Pogled pošilja posodobitve nadzoru, nadzor posodablja modele in model posodablja poglede. V praksi v Yii to pomeni, da so v projektu objekti, ki definirajo neko strukturo, ki jo želimo shraniti v podatkovno shrambo, in Yii omogoči, da lahko programsko ustvarjamo, popravljamo ali brišemo objekte. Ogrodje samodejno poskrbi, da se podatki zapišejo v podatkovno shrambo. Ta način preslikave objektov v podatkovno shrambo in iz nje se imenuje ORM (*Object-relational mapping*) in je bistvena funkcionalnost, potrebna za ločitev aplikacijske plasti od vrste podatkovne shrambe, kar omogoča abstrakcijo komunikacije s podatkovno shrambo oz.



Slika 3.2: MVC-arhitektura aplikacijskega sloja.

zakrivanje vrste uporabljene podatkovne shrambe. Yii podpira več podatkovnih shramb, in sicer MySQL, SQLite, PostgreSQL, Microsoft SQL Server in Oracle.

Naslednja izmed pomembnejših funkcionalnosti ogrodja je avtomatsko sestavljanje programske kode CRUD (*Create Retrieve Update Delete*). Yii-ogrodje omogoča, da razvijalec pripravi podatkovni model v shrambi, iz atributov entitet in relacij med entitetami nato Yii pripravi kodo, ki omogoči osnovne funkcionalnosti za ustvarjanje, prikazovanje, posodabljanje in brisanje posameznih podatkovnih struktur. Ta funkcionalnost razvijalcem omogoča, da takoj po pripravljeni podatkovni shemi že dobijo osnovne obrazce, tabele, iskanja, izpise itd. Razvijalci imajo tako hitro pripravljene osnovne dele aplikacije, kar pomeni, da jim ni treba izgubljati časa s pisanjem vedno istih enostavnih stvari, ampak se lahko ukvarjajo z razvojem funkcionalnosti sistema.

V sistemih, kjer se večkrat dostopa do istih podatkov in kjer se ista programska koda izvaja večkrat, njen rezultat pa je v večini primerov enak, se za pospeševanje delovanja spletnih strani uporabi sistem za predpomnjenje. S

predpomjenjem se posamezni podatki in rezultati izvajanja programske kode predpomni v shrambi za predpomnjenje, in ko se ponovno pojavi zahteva po posameznem podatku, je le-tega mogoče prebrati neposredno iz shrambe za predpomnjenje, kar je časovno veliko hitrejša in procesorsko manj zahtevna operacija kot ponovno izvajanje branja iz podatkovne shrambe oz. ponovno izvajanje programske kode. Yii ima odlično podporo za predpomnjenje (*caching*, večkrat dostopane podatkovne objekte, obdelane predloge itd. je mogoče obdelane shraniti v shrambo za predpomnjenje). Ogrodje ima implementiran abstrakten dostop do sistema za predpomnjenje, tako da se lahko uporabi vrsta različnih sistemov shramb za predpomnjenje (*memcache, APC cache, XCacge, EAccelerator, RedisCache, DBCache, ZendCache, WinCache, FileCache*). Predpomnjenje je mogoče implementirati za nek določen čas, tako da določimo pogoje, do kdaj je nekaj predpomnjeno, ali dokler se ne spremeni nek objekt, datoteka ali zapis v podatkovni shrambi.

Poleg naštetih funkcionalnosti ogrodja so zelo pomembne še podpora za internacionalizacijo in lokalizacijo, kar omogoča enostavno prevajanje in podporo za izpis datumov itd. Podpora za obravnavanje napak omogoča delovanje aplikacije z dodatnimi dnevniškimi zapisi, kar omogoča lažje razhroščevanje. Poleg naštetega omogoča ogrodje še veliko podpore za varnost in zaščito pred napadi XSS, CSRF, spremembe piškotkov (kriptiranje vsebine piškotkov, HMAC-zaščita piškotkov) [55] itd.

Z Yii ogrodjem je narejenih veliko precej uspešnih spletnih strani in projektov, kot na primer: Zurmo (Open Source CRM), Noisey.com, Stay.com, VICE.com, RealSelf.com, Hotellweb.no, Nutritionix.com ...

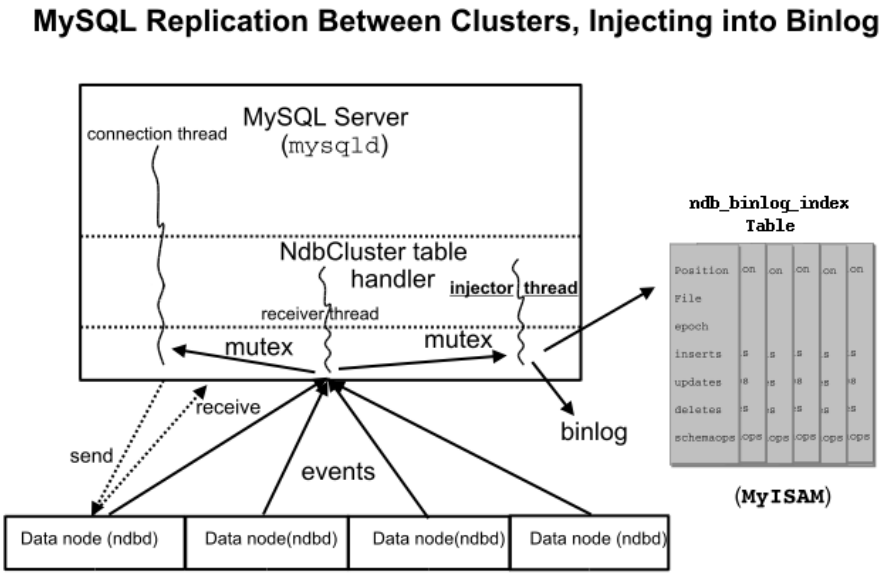
3.7 MySQL Cluster

Družina podatkovnih baz MySQL je zelo široka in omogoča praktično popolno podatkovno skladnost med posameznimi različicami strežnika MySQL. S tem so omogočili, da lahko v razvojnem okolju uporabljamo standardno obliko MySQL-strežnika, ki je praktično vključena v vsako osnovno razvojno

orodje XAMPP (LAMPP) [1], kar precej poenostavi pripravo razvojnega okolja za razvijalce sistema. Vsa razlika med uporabo standardnega MySQL-strežnika in gručnega strežnika je v nastavljenem podatkovnem pogonu posamezne tabele: v lokalnem razvojnem okolju se uporablja InnoDB, v produkcijskem gručnem sistemu pa ndbcluster, to pomeni, da je mogoče strukturo in podatke iz ene različice strežnika prestavljati praktično z eno samo menjavo nastavitve tabele v podatkovni bazi. Druga najpomembnejša stvar družine podatkovnih baz MySQL je, da sta vmesnik SQL [39] in način dela s tabelami, strukturami in podatki praktično enaka med vsemi različicami, kar pomeni, da uporaba druge različice strežnika ne predstavlja popolnoma nobene spremembe v programski kodi aplikacijskega sloja.

MySQL Cluster je tehnologija, ki omogoča gručenje nič-skupnega (*shared-nothing*) [47] za podatkovni sistem in orodja MySQL. Arhitektura shared-nothing je arhitektura za porazdeljeno računanje, v kateri je vsako vozlišče sistema neodvisno eno od drugega in samozadostno, kar za celoten sistem pomeni, da v sistemu ni nobene kritične točke, od katere bi bil celoten sistem odvisen in ki bi lahko povzročila izpad delovanja sistema kot celote. MySQL Cluster je bil narejen z namenom, da omogoči visoko zanesljivost in veliko prepustnost podatkov z majhno latenco ter kar se le da linearno povečevanje zmogljivosti. MySQL Cluster je implementiran skozi podatkovni pogon NDBCLUSTER, kjer NDB kratica pomeni mrežna podatkovna baza (*Network Database*). Podatkovni sistem omogoča avtomatsko uporabo particij (particija je delitev logične baze podatkov ali njenih sestavnih elementov v ločene samostojne dele) za izboljšavo bralnih in pisalnih operacij na različni strojni opremi. Dostop do podatkov je mogoč preko vmesnikov SQL in NoSQL API. Za potrebe tekmovalnega sistema smo uporabili SQL API [36].

Replikacija oz. podvojitev podatkov v podatkovni bazi poteka sinhrono skozi dvofazni potrditveni mehanizem (glej slika 3.3, dogodki *send, recieve*, nadzor preko *mutex*-a), kar zagotavlja, da so podatki zapisani na več vozlišč ob prejetem potrdilu o zapisu podatkov. Vsaj dve kopiji podatkov sta

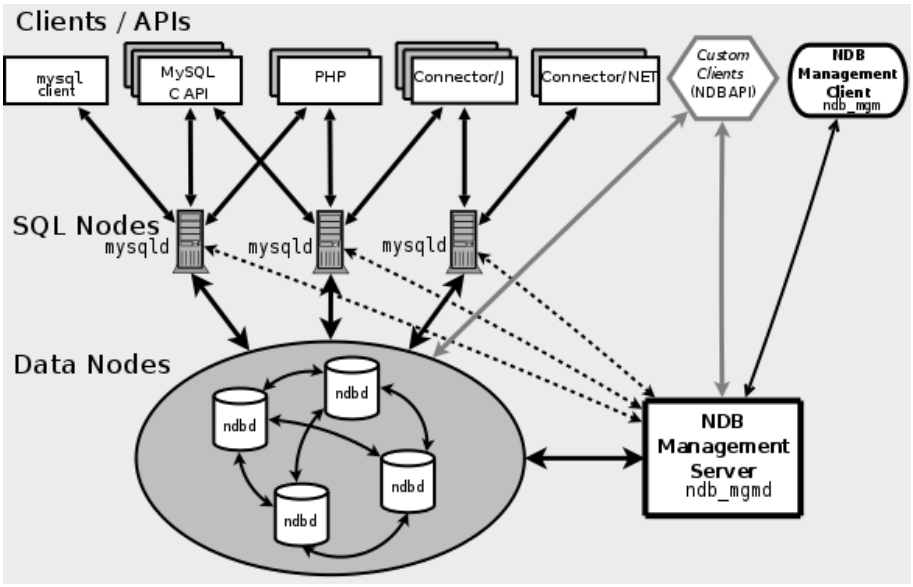


Slika 3.3: Komunikacija med SQL-vozliščem in podatkovnimi vozlišči, prikazan postopek podvojitve podatka med strežniki v gruči [38].

potrebni za vzdrževanje zagotovila, da so podatki zapisani. Sistem avtomatsko pripravi skupine vozlišč na podlagi števila replikacij, ki se ga nastavi v konfiguraciji. Vse posodobitve so sinhrono replicirane med vsemi vozlišči znotraj skupine vozlišč, s tem se prepreči izguba podatkov in omogoči hiter preklop med vozlišči v primeru izpadov posameznih vozlišč. MySQL Cluster omogoča tudi asinhrono replikacijo. Tako vrsto replikacije imenujejo tudi geo-replikacija, ker se z njo običajno replicirajo gruče v različnih podatkovnih centrih, različnih fizičnih lokacijah, za primere obnovitev podatkov v primeru nepričakovanih dogodkov (izpada elektrike, poplave, naravne nesreče ...). Asinhrono replikacijo se lahko uporabi tudi zaradi zmanjšanja komunikacijske latence med uporabniki podatkov in lokacijo podatkov (podatki bližje uporabniku) [37].

MySQL Cluster omogoča, da je količina podatkov v gruči lahko večja od velikosti diskovnega polja posameznega vozlišča v sistemu. MySQL Cluster

drži vse indekse stolpcev tabel v porazdeljenem pomnilniku. Vse neindeksirane stolpce pa sistem prav tako lahko drži v porazdeljenem pomnilniku ali pa na disku s predpomnjenimi stranmi v pomnilniku. Shranjevanje neindeksiranih stolpcev tabel na disku omogoči, da lahko MySQL Cluster shrani podatke, večje od celotnega pomnilnika vseh vozlišč v gruči. V našem primeru smo želeli preveriti popolne zmogljivosti sistema, zato smo virtualnemu strežniku dodelili dovolj pomnilnika, da je lahko vse podatke hranil v njem. Kot sekundarni način shranjevanja podatkov in informacij sistem ves čas asinhrono piše tako imenovani dnevniški zapis ,,naredi ponovno“ (*ReDo, binary log*, glej sliko 3.3 *binlog*), v katerega si zapisuje vse spremembe podatkov, kar omogoča dodatno zaščito podatkov v primeru, da pride do popolnega izklopa vseh vozlišč v gruči. Zaradi asinhronega zapisa je mogoče, da se v primeru popolnega izpada nekaj transakcij/sprememb podatkovne baze lahko izgubi [34].



Slika 3.4: Prikaz postavitve gruče MySQL Cluster in načinov komunikacije, ki so omogočeni preko SQL-vozlišč [36].

Implementacija celotnega sistema MySQL Cluster na sliki 3.4 je sestavljena iz treh osnovnih vozlišč oz. procesov, in sicer: podatkovno vozlišče

(`ndbd / ndbmtd`, na sliki 3.4 spodaj levo), nadzorno vozlišče (`ndb_mgmd`, na sliki 3.4 spodaj desno) in aplikacijsko oz. SQL-vozlišče (`mysqld`, na sliki 3.4 na sredini nad podatkovnimi vozlišči).

Podatkovna vozlišča shranjujejo podatke, tabele so avtomatsko porazdeljene med vsa podatkovna vozlišča, vsi postopki porazdelitve, replikacije, preklopa v primeru izpada in ponovne vzpostavitve vozlišča so popolnoma prikriti drugim vrstam vozlišč v sistemu.

Nadzorno vozlišče upravlja s stanjem gruče, skrbi za nastavljanje in spremljanje stanja vseh vozlišč v gruči. Načeloma se ta vozlišča potrebujejo samo ob zagonu, ponovnem zagonu vozlišč in izpadih. Običajno so ta vozlišča arbitri, to pomeni, da ta vozlišča odločajo o trenutnem glavnem (*master*) podatkovnem vozlišču.

Aplikacijska oz. SQL-vozlišča so vozlišča, ki se povezujejo na podatkovna vozlišča in od tam pridobivajo oz. shranjujejo podatke. Uporabljajo se za zakrivanje gruče aplikacijam, ki uporabljajo MySQL v osnovni različici, ker je uporaba gručne različice popolnoma skladna z vmesnikom negručnega MySQL-vmesnika. To vozlišče načeloma ni potrebno, ker obstajajo tudi neposredni vmesniki NDB API, ki omogočajo da se do podatkov dostopa neposredno preko vmesnika C++ API. Vendar je za potrebe tekmovalnega sistema boljši in lažji način dostopa do podatkov preko standardnega SQL-vmesnika, ker je enak kot v standardni obliki MySQL-strežnika.

Poglavje 4

Opis rešitve

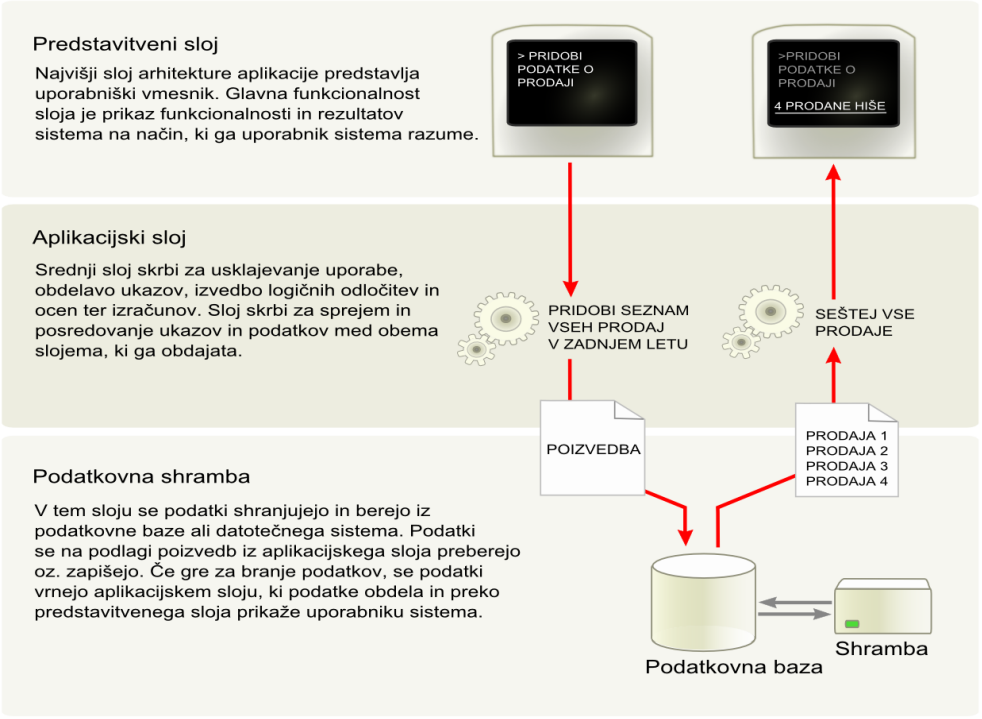
4.1 Abstraktna 3-slojna arhitektura

Večslojna arhitektura sistema omogoča, da lahko razvijalci sistemov ločeno razvijajo posamezen sloj, tako da obstajajo različni strokovnjaki za vsak sloj. Tak model razvoja programske opreme je zelo praktičen, ker razvijalce usmeri pri razvoju in programiranju posameznih logičnih kosov programja, ki jih je mogoče Ločenost posamičnih slojev omogoča enostavno dodajanje in spreminjanje funkcionalnosti na posameznem sloju, tako razvijalcem ni treba več čas razvoja razmišljati o celotnem sistemu. Ker je sistem zasnovan tako, da so sloji med seboj logično ločeni, je mogoče zamenjati vsak sloj posebej, recimo kot nova implementacija storitve, druga tehnologija, platforma, drugačen način shranjevanja podatkov itd. V večslojnih sistemih je hierarhija slojev zelo pomembna, noben sloj ne more preskočiti neposrednega soseda, v praksi to pomeni, da odjemalec ne more nikoli neposredno dostopati do podatkovne shrambe, ampak mora komunikacija vedno potekati preko aplikacijskega sloja.

Trislojna arhitektura (opisana tudi na sliki 4.1[1]) je arhitektura odjemalec-strežnik, ki ima ločene tri glavne, med seboj neodvisne sloje oz. module, in

[1]Slika po `http://upload.wikimedia.org/wikipedia/commons/5/51/-Overview_of_a_three-tier_application_vectorVersion.svg`.

sicer uporabniški vmesnik (predstavitveni sloj), funkcijsko-procesno logiko (aplikacijski sloj) in podatkovno shrambo. Popolnoma nič neobičajnega ni, da je vsak sloj lahko na popolnoma drugačni platformi. Trislojno arhitekturo je razvil John. J. Donovan v svojem podjetju Open Enviroment Corporation (OEC) okoli leta 1992 v Cambridgeu v zvezni državi Massachusetts [16].



Slika 4.1: 3-slojna arhitektura.

4.1.1 Predstavitveni sloj oz. uporabniški vmesnik

To je najvišji nivo arhitekture sistema, pogosto ga imenujejo tudi predstavitveni sloj, ker se ta sloj uporablja predvsem za prikaz informacij. V sistemih, ki delujejo na spletu, je pri tem sloju večinoma govora o odjemalcu oz. spletnem brskalniku in načinu prikaza podatkov s pomočjo spletnih tehnologij HTML, CSS in JavaScript.

Ta nivo poskrbi za komunikacijo z aplikacijskim slojem, predvsem za to, da pridobljene podatke s srednjega sloja uporabniku prikaže čim bolj struk-

turirano in pregledno. V ta nivo se dodajo tudi različna preverjanja pravilnosti vnesenih podatkov na uporabniškem vmesniku na strani odjemalca, preden se podatki pošljejo v aplikacijski sloj, kar omogoča znatno zmanjšanje nepotrebne komunikacije s strežniki (manjša obremenjenost strežnikov). V današnjem mobilnem svetu, kjer je hitrost interneta omejena, postaja to še toliko bolj pomembno. Kljub vsem preverjanjem na tem nivoju se je treba zavedati, da podatki, ki pridejo iz tega sloja, nikoli ne morejo biti popolnoma pravilni, zato je treba vse vhodne podatke na strani aplikacijskega sloja vedno preveriti še enkrat.

Na tem nivoju je velik poudarek na uporabniški izkušnji s sistemom, predvsem je želja po tem, da so uporabniki zadovoljni z videzom aplikacije, da so gumbi na pravilnih mestih, da se na uporabniškem vmesniku uporabnik znajde in da lahko prične uporabljati aplikacijo brez dodatnih navodil za uporabo.

Porazdeljenost tega sloja je trivialna, ko se sistem povečuje, je odjemalcev vedno več.

4.1.2 Vmesni oz. aplikacijski sloj

Aplikacijski sloj je najpomembnejši sloj v celotni arhitekturi sistema, saj skrbi za tp, da se v celotnem sistemu nekaj izračuna, obdela, shrani in prikaže. Ta del arhitekture je pomemben predvsem zato, ker mora skrbeti za komunikacijo med dvema slojema sistema, in sicer med odjemalcem in podatkovno shrambo.

Med delovanjem sistema mora aplikacijski sloj skrbeti za pravilne vnose odjemalcev. To v praksi pomeni, da aplikacijski sloj najprej preveri pravice uporabnika, namreč ali ta sploh lahko izvede določeno akcijo, nato preveri vnešene podatke, in če so podatki pravilni in je uporabnik avtoriziran, aplikacijski sloj podatke zapiše v podatkovno shrambo. Ter obratno, ko želi odjemalec pridobiti nek podatek iz sistema oz. podatkovne shrambe, se najprej preveri, ali ima odjemalec pravico pridobiti nek podatek, in če jo ima, se podatek pridobi iz podatkovne shrambe. Pridobljene podatke se zapakira

na način, ki ga odjemalec razume, nato se zapakirane podatke posreduje kot odgovor na odjemalčevo zahtevo.

Pri sistemih, ki delujejo na spletu, se o tem sloju govori predvsem kot o tehnologijah, ki se izvajajo na strežnikih in ki omogočajo izgradnjo dinamičnih vsebin, bolj poznane so tehnologije PHP, Java EE, Ruby on Rails, ASP.NET, Perl.

Pri aplikacijskem sloju je zanimivo, da se večinoma deli še na več slojev, zelo znana je arhitekturna razdelitev aplikacijskega sloja na MVC (opisana v poglavju 3.6).

Ena izmed pomembnejših nalog tega sloja sistema je identifikacija uporabnikov, sistem mora zaradi svoje lokacije med dvema plastema ustrezno poskrbeti, da so v sistemu odjemalci ustrezno označeni, da se podatki za njih ustrezno shranijo in da ne prihaja do zlorab sistema, nepooblaščenega dostopa do podatkov itd. Pri spletnih sistemih se to običajno rešuje z uporabo sej oz. piškotkov. Podatki o sejah in piškotkih se za potrebe porazdeljenega aplikacijskega sloja zapisujejo v podatkovno shrambo, ki poskrbi, da so informacije dostopne vsem strežnikom aplikacijskega sloja. Piškotki, zapisani v podatkovni bazi, so najboljši način identifikacije, saj zagotavljajo, da uporabnik ne more spremeniti shranjenih podatkov v seji. Obstaja pa tudi manj varna različica uporabe piškotkov, in sicer hranjenje in spreminjanje piškotkov na strani uporabnika, vendar za potrebe boljše in zanesljivejše varnosti ta način ni priporočljiv, ker obstaja možnost, da na strani uporabnika nekdo spremeni vrednost piškotka in tako se strežnik lahko začne drugače obnašati do tega uporabnika. V primeru, da bi v vrednost piškotka zapisali ID prijavljenega uporabnika in bi strežnik samo preverjal, ali je ID prijavljenega uporabnika nastavljen, bi lahko uporabnik lokalno spremenil v piškotku zapisani ID na drug ID uporabnika in sistem bi mislil, da je to drug uporabnik.

Dodatna zaščita, poleg piškotkov in sej, ki jih je mogoče relativno enostavno skopirati, je uporaba uporabniških kvalificiranih digitalnih potrdil, ki skrbijo, da je uporabnik identificiran na podlagi varno vzpostavljene po-

vezave s strežnikom. Za identifikacijo uporabnika se uporabljajo podatki o uporabniku, zapisani v digitalnem potrdilu, podatkom pa zaupamo, ker zaupamo izdajatelju digitalnih potrdil. Uporaba kvalificiranih digitalnih potrdil se običajno uporablja za povezovanje na spletne bančne sisteme in različne informacijske sisteme, ki zahtevajo visok nivo varnosti in zaupanja. V primeru tekmovalnega sistema je tak način nepraktičen, ker tekmovalci nimajo svojih lastnih digitalnih potrdil, ki bi jih lahko uporabili za identifikacijo.

Porazdeljenost tega sloja je zelo zanimivo področje, saj vsak sistem stremi k temu, da bi lahko imel čim večje število uporabnikov in da bi bili njegovi uporabniki s storitvijo sistema čim bolj zadovoljni. Hkrati pa je ta sloj procesno in pomnilniško zahteven in vsak lastnik sistema stremi k temu, da bi bila poraba čim manjša in cenejša. Na tem sloju se pojavi problem, kako postaviti sistem, da bo mogoče programsko opremo tega sloja izvajati na večih fizičnih strežnikih, s čimer se poveča zmogljivost in zanesljivost celotnega sistema. Porazdeljenost aplikacijskega sloja se rešuje z zagonom storitev na različnih strežnikih, podatke pa se bere oz. zapisuje v neko skupno podatkovno shrambo, ki je dosegljiva vsem strežnikom. S skupno podatkovno shrambo se rešuje problem hranjenja stanja, ko prihajajo zahtevki uporabnikov na različne strežnike aplikacijskega sloja.

4.1.3 Podatkovna shramba

Najnižji nivo arhitekture je podatkovna shramba oz. enostavno rečeno, podatki. Podatki so pomembni v vsakem sistemu, saj omogočajo, da se lahko nekaj obdeluje in prikazuje. Na tem sloju se nahajajo podatkovni strežniki, ki so običajno sestavljeni iz podatkovnih sistemov in sistemov za upravljanje s podatkovnimi sistemi. Naloga sloja je enostavna, omogočati mora le, da se podatek lahko shrani in nato prebere. Ta sloj podatke običajno hrani čimbolj neodvisno od aplikacijskega sloja, kar omogoča, da je podatkovna baza veliko bolj povečljiva in zmogljivejša.

Pri spletnih sistemih se pri tem nivoju uporabljajo podatkovni sistemi dveh tipov, SQL in noSQL. Glavna razlika med njima je način shranjeva-

nja podatkov, in sicer ima tip SQL podatke zapisane v tabeli po vnaprej določeni shemi, posamezna polja v tabeli so lahko v relaciji z drugimi tabelami, NoSQL pa ima podatke zapisane kot objekte, pri čemer ima vsak objekt lahko drugačno strukturo. NoSQL ne omogoča relacij med objekti in nima vnaprej določene sheme za zapis posameznega objekta. Druga glavna razlika je v vrsti povečljivosti posameznega tipa. SQL se povečuje vertikalno (za večjo zmogljivost sistema je potrebna večja moč strežnika, boljši procesor, več spomina na posameznem vozlišču), kar pomeni, da mora biti zmogljivost posameznega strežnika premosorazmerna glede na povečano število zahtevkov oz. povečano število podatkov. Načeloma je mogoče SQL porazdeliti na več strežnikov, vendar to zahteva veliko dodatnega inženiringa (narejeno v tem projektu). NoSQL se povečuje horizontalno (za večjo zmogljivost sistema se doda enako zmogljive strežnike), potreba po dodatni kapaciteti preprosto pomeni dodatne strežnike, podatkovna baza se po potrebi avtomatsko porazdeli na nove strežnike.

Velika razlika med sistemoma je tudi v zapisu podatkov, SQL-podatki imajo strukturo, noSQL-podatki je nimajo. Zaradi razlike v zapisu podatkov je nastala tudi razlika v načinu dostopa do podatkov, shranjenih v shrambi, in sicer podatkovne baze SQL temeljijo na relacijski algebri, ki je implementirana z jezikom SQL. NoSQL poizvedbe so poizvedbe tipa ključ-vsebina, kar pomeni, da za dostop do podatka nujno potrebujemo ključ, drugače do podatka ne moremo dostopati, za razliko od SQL tukaj ni mogoče narediti poizvedb po atributih znotraj vsebine posamezne entitete.

Zaradi uvedbe strukture pri SQL, ki je pripomogla k lažjim in kompleksnejšim poizvedbam med podatki, je nastala dodatna skrb za konsistenco, ki je povzročila dodatne potrebe po procesorski zmogljivosti in spominu. Z rastjo količine podatkov so se povečevale tudi potrebe po strojni opremi. Osnove podatkovne baze SQL so nastale v letih 1970, ko so se kazale potrebe po boljši organiziranosti podatkov in lažjih poizvedbah. V letu 2000, ko so se začele pojavljati hitro rastoče spletne storitve, so nastale podatkovne baze NoSQL, ki rešujejo potrebe po boljši povečljivosti, replikaciji in nestrukturi-

rani podatkovni shrambi [45].

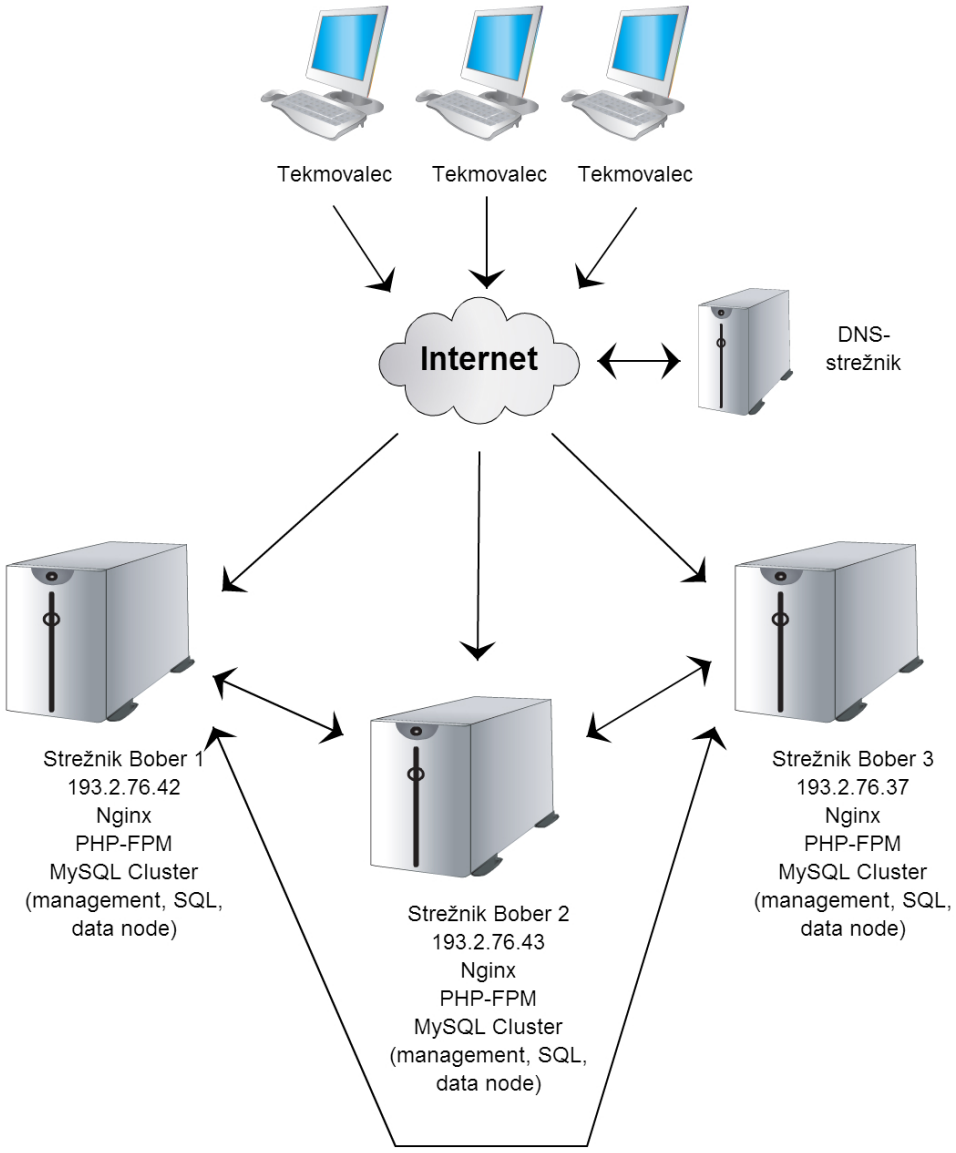
Primeri pogosteje uporabljenih relacijskih podatkovnih baz so dandanes: MySQL, PostgreSQL, Microsoft SQL Server, Oracle, SQLite, MariaDB, medtem ko so primeri pogosteje uporabljenih podatkovnih baz NoSQL: mongoDB, Cassandra, HBase, Neo4j, Hypertable, DynamoDB, Google Datastore, Amazon SimpleDB in Cloudera.

Porazdeljenost tega nivoja je zelo kompleksna in stremi predvsem k načinu, kako podatke shranjevati, da se doseže večjo zmogljivost in kapaciteto. Več o tem je opisano v poglavju 3.2.

4.2 Postavitev strežnikov v oblaku

Tekmovalčev brskalnik (rdeč krog na sliki 4.3) se je poskusil povezati s strežnikom na naslovu bober.acm.si. V primeru, da spletni brskalnik ne pozna IP-naslova strežnika na naslovu bober.acm.si, brskalnik posreduje zahtevo za razreševanje DNS-zapisa lokalnem razreševalniku DNS-domen na tekmovalčevem računalniku, če le-ta še vedno ne pozna IP-naslova, pošlje DNS A zahtevo našemu DNS-strežniku (na sliki 4.2 desno, na shemi poteka komunikacije na sliki 4.3 zgoraj desno). DNS-strežnik ima v svoji konfiguraciji nastavljene 3 zapise DNS A za domeno bober.acm.si. DNS-strežnik po algoritmu Round-Robin [6] (na sliki 4.3 zgoraj) vsakemu tekmovalcu vrne en IP-naslov enega izmed strežnikov Bober in s tem zagotovi enakomerno razporeditev uporabnikov na vsakega izmed strežnikov (na sliki 4.2 Strežnik Bober 1, 2 in 3).

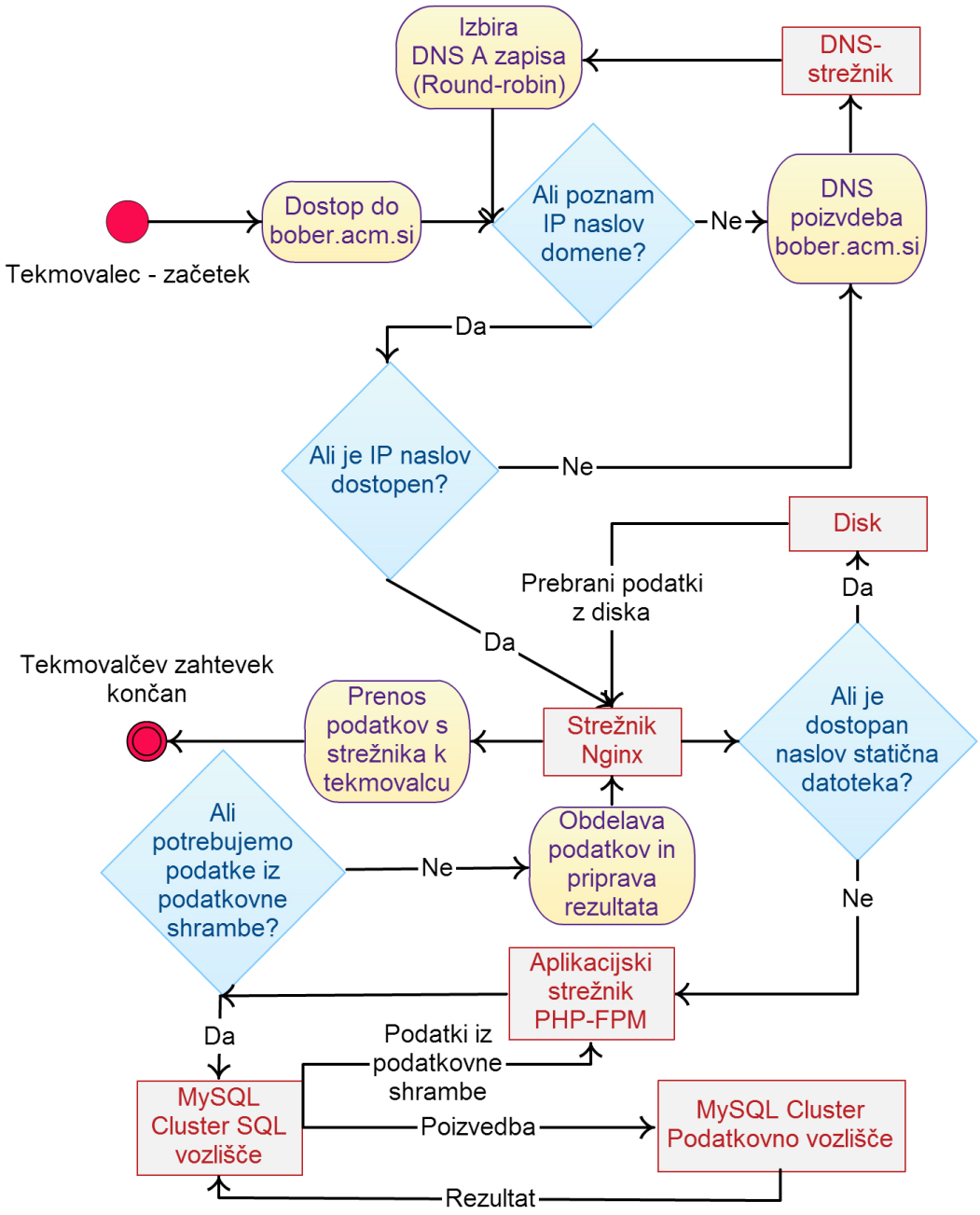
Ko ima brskalnik IP-naslov strežnika za bober.acm.si, naredi HTTP-zahtevek na strežnik. V primeru, da IP-naslov na HTTP-vratih ni dostopen, se ponovi postopek pridobivanja IP-naslova (na sliki 4.3 zgoraj desno). Na strežniku HTTP-zahtevek sprejme strežnik Nginx (na sliki 4.2 spodaj, ob vsakem strežniku so navedene vse storitve, ki so na voljo, na sliki 4.3 na sredini).



Slika 4.2: Shema postavitve strežnikov tekmovalnega sistema Bober.

Nginx razpakira zahtevek in statične datoteke (JavaScript, CSS, slike) neposredno prebere z diska in vrne uporabniku, zahtevke, ki pa potrebujejo obdelavo aplikacijskega sloja, pošlje naprej na aplikacijski strežnik PHP-FPM (na sliki 4.3 spodaj desno). Na aplikacijskem strežniku se izvede programska koda tekmovalnega sistema, naredijo se morebitne poizvedbe v podatkovno shrambo MySQL Cluster (na sliki 4.3 spodaj levo), nato se rezultat vrne

strežniku Nginx in le-ta nato rezultat posreduje nazaj uporabniku, uporabnikov zahtevek se s tem zaključi (na sliki 4.3 na sredini levo).



Slika 4.3: Shema komunikacije med odjemalcem in posameznimi strežniki tekmovalnega sistema.

Vsak aplikacijski strežnik PHP-FPM deluje neodvisno od ostalih strežnikov, strežniki med seboj niso povezani. Za podatkovno shrambo je uporabljen MySQL Cluster, strežniki so na nivoju podatkovne shrambe med seboj povezani. MySQL Cluster ima v našem primeru na vsakem strežniku zagnane tri storitve (v osnovi je lahko vsaka od teh storitev na ločenem strežniku), in sicer vozlišče za upravljanje (*management node*), SQL-vozlišče (*SQL node*) in podatkovno vozlišče (*data node*).

4.3 Implementacija 3-slojne arhitekture

Arhitektura je bila razdeljena na tri sloje, in sicer so to spletni odjemalec oz. brskalnik (HTML, JavaScript, CSS), aplikacijsko oz. vmesno plast (PHP-spletna aplikacija) in podatkovno shrambo (MySQL Cluster). Za tako implementacijo smo se odločili, ker želimo doseči neodvisnost posameznih slojev od celotnega sistema, kar pomeni, da se bomo lahko vsakemu sloju posebej posvetili in ga ločeno optimizirali in prilagodili za doseganje visoke zmogljivosti, ki jo zahtevajo kvantitativne zahteve sistema. 3-slojna arhitektura je bila izbrana, ker smo po več prebranih virih ugotovili, da je to dobra arhitektura za sisteme, kjer želimo imeti veliko povečljivost in zmogljivost [25, 31].

4.3.1 Predstavitveni sloj oz. uporabniški vmesnik

Zahteve uporabniškega vmesnika izhajajo iz funkcionalnih zahtev sistema. Najpomembnejša in najbolj problematična zahteva sistema je pripraviti uporabniški vmesnik, s katerim bomo lahko dosegali veliko prožnost in razširljivost. Za ustrezno razširljivost tekmovalne platforme smo za osnovo uporabili interaktivne naloge (opisane v poglavju 1.4.3). Prožnost smo dosegli z izdelavo uporabniškega vmesnika, katerega programska koda se izvaja predvsem v spletnem brskalniku odjemalca, in si tako zagotovili razbremenitev strežnikov.

V prvem letu delovanja smo se za namene tekmovalnih nalog odločili implementirati poenostavljeni način priprave nalog. Poenostavljene interak-

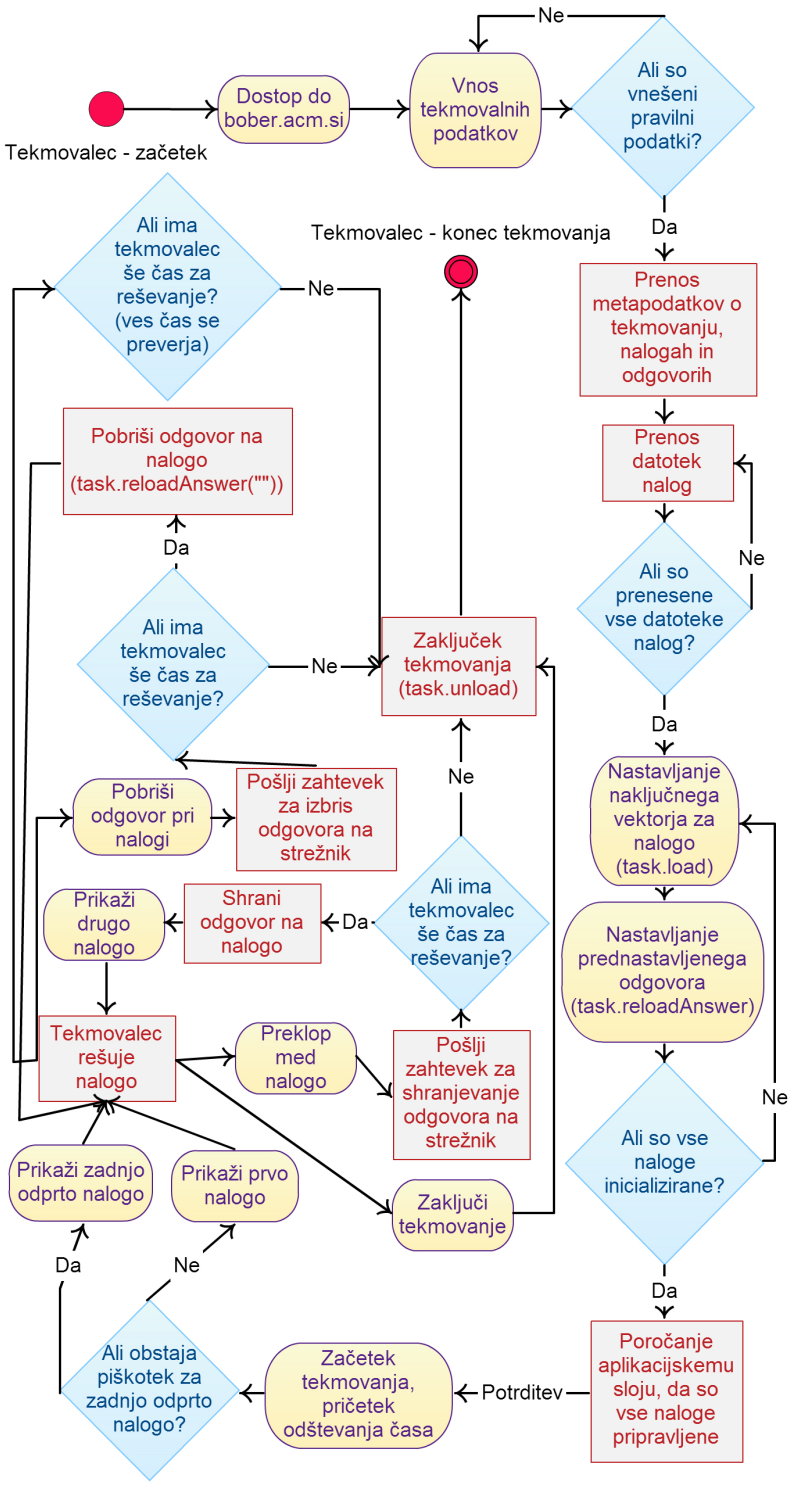
tivne naloge so tiste, pri katerih je edina interaktivnost, ki jo v svojem programskem delu omogočajo naloge, da znajo s pomočjo vhodnega naključnega vektorja premešati vnaprej določene odgovore naloge, kar omogoča, da tekmovalci na tekmovanju nimajo popolnoma identičnih nalog.

Vzporedno z razvojem tekmovalne platforme in priprave ideje interaktivnih nalog se je začela pripravljati tudi spletna aplikacija za izgradnjo nalog v okviru magistrske naloge Grega Pompeta. Sistem za izgradnjo nalog podpira pripravo besedil in slik nalog, omogoča nastavitev odgovorov in izvoz v formatu za interaktivne naloge. Naloge lahko urejajo različni uporabniki. Možno je tudi prevajati naloge iz enega v drug jezik. Skozi ves postopek priprave nalog se beleži zgodovina sprememb naloge, tako da je mogoče slediti spremembam in jih v primeru določene nezaželene spremembe povrniti v prejšnje stanje.

Na podlagi zahtev interaktivnih nalog in tekmovalne platforme smo se odločili, da bomo za uporabniški vmesnik uporabili tehnologije HTML5, CSS in JavaScript. S pomočjo teh tehnologij je bilo mogoče učinkovito implementirati delovanje nalog v spletnih brskalnikih. Tekmovalna platforma je sposobna s pomočjo programskega jezika JavaScript komunicirati preko standardnih, vnaprej definiranih klicev z interaktivnimi nalogami.

Komunikaciji med vmesnim slojem in uporabniškim vmesnikov smo posvetili veliko časa, saj smo komunikacijo želeli kar se da zmanjšati in tako povečati zmogljivost posameznega strežnika v času tekmovanja, ko lahko pričakujemo tudi do 10.000 hkratnih tekmovalcev, pri čemer vsaka dodatna komunikacija pomeni dodatno obremenitev strežnikov.

Uporabniški vmesnik za tekmovanje (zahteve opisane v poglavju 1.4) je optimiziran, način in potek tekmovanja, ki mu je mogoče slediti tudi na sliki 4.4, opisujem na tem mestu. Pri tekmovanju tekmovalec obišče spletno stran tekmovanja, kjer izpolni prijavne podatke, nato je preusmerjen na stran, na kateri se s strežnika prenesejo potrebne datoteke tekmovalne platforme CSS in JavaScript.

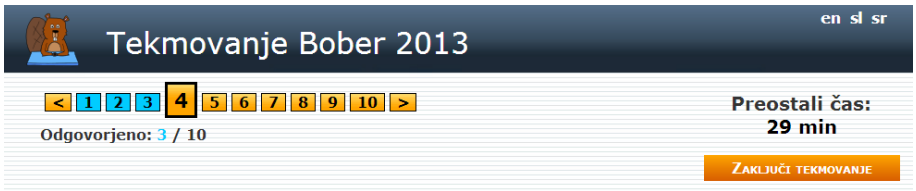


Slika 4.4: Potek tekmovanja – shema komunikacije tekmovalne platforme z interaktivno nalogo in strežnikom.

V datotekah JavaScript je implementirana logika, ki na strani odjemalca izvede programsko kodo in naloži tekmovalne naloge s strežnika, ki jih poda aplikacijski sloj tekmovalnega sistema na podlagi pripravljenega tekmovanja (metapodatki nalog, podatki o virih, vhodni naključni vektor, predhodno izbrani odgovor) in jih lokalno predpomni v brskalnikov predpomnilnik (programska koda pregleda vse metapodatke nalog in vnaprej sproži prenos virov nalog, tako da takrat, ko uporabnik želi videti nalogo, ni treba več prenašati podatkov s strežnikov). Ko so vse potrebne datoteke tekmovalnih nalog naložene, tekmovalna platforma JavaScript nad tekmovalnimi nalogami izvede ukaz, da se nastavi prednastavljeni vrstni red odgovorov (tekmovalna platforma za istega tekmovalca vedno poda enako vhodno, sicer naključno število, in interaktivna naloga na podlagi le-tega v funkciji `task.load` nastavi enako vrsto odgovorov) in prednastavljeni odgovor, če je uporabnik pred tem že reševal tekmovanje in je morda zaradi nedelovanja ali nenamerno naredil osvežitev spletne strani. Ko so vsi odgovori nastavljeni, koda na strežnik sporoči, da je tekmovanje lokalno pripravljeno za tekmovanje, in ko aplikacijska plast prejme ukaz, si zapiše čas pričetka tekmovanja, JavaScript kodi sporoči, da se je tekmovanje za tekmovalca pričelo, v tem trenutku programska koda na strani odjemalca nastavi uro, ki odšteva minute do konca tekmovanja ter prikaže prvo nalogo (oz. nalogo, na kateri je uporabnik ostal pred osvežitvijo spletne strani, ob vsakem premiku med nalogami se lokalno na uporabnikovem računalniku nastavi piškotek, v katerem se shrani zadnja odprta naloga). Nato ima tekmovalec čas, da rešuje naloge, ko preostane do konca tekmovanja samo še pet minut, tekmovalna platforma JavaScript spremeni barvo preostanka časa v rdeče in s tem opozori tekmovalca, da mu bo kmalu zmanjkalo časa.

Pri prehodu med nalogami se na strežnik vsakič sporoči, kakšen odgovor je bil izbran ob prehodu. Vmesnik omogoča, da lahko pri vsaki nalogi uporabnik odstrani trenutno izbrani odgovor, kar je pomembno, saj napačen odgovor za tekmovalca pomeni negativne točke. Na vmesniku na sliki 4.5 se pri prikazu seznama nalog glede na to, ali je odgovor izbran ali ne, naloge barvno

označujejo (svetlo modra – naloga odgovorjena, rumena barva – naloga še ni odgovorjena), tako da ima tekmovalec ves čas pregled nad tem, koliko nalog mu je še ostalo za reševanje.



Slika 4.5: Grafični vmesnik tekmovalca med tekmovanjem.

V primeru, da poteče čas tekmovanja (na sliki 4.5 desno), koda JavaScript izpiše, da je tekmovalni čas potekel, in tekmovalca preusmeri nazaj na začetno stran, na kateri lahko naslednji tekmovalec že prične s tekmovanjem. V primeru, da tekmovalec želi zaključiti tekmovanje pred koncem časa tekmovanja, uporabniški vmesnik prikaže opozorilo, da po zaključku tekmovanja ne bo več mogoče reševati. V primeru, da ob kliku na gumb `Zaključi tekmovanje` na sliki 4.5 desno spodaj še niso rešene vse naloge, uporabniški vmesnik izpiše opozorilo s podatkom o številu nerešenih nalog in uporabniku ponudi možnost odločitve, ali resnično želi končati s tekmovanjem ali morda želi nadaljevati reševanje.

Uporabniški vmesnik za administracijo je popolnoma standarden in enostaven. Za njegovo izgradnjo smo uporabili vgrajene generatorje vmesnika v vmesnem sloju platforme Yii in se mu bolj podrobno nismo posvečali. Administrativni del vmesnika je ostal pod popolno kontrolo vmesnega aplikacijskega sloja.

4.3.2 Vmesni sloj/aplikacijski sloj

Aplikacijski sloj je najpomembnejši sloj celotnega sistema. V tem sloju smo morali implementirati kompletno logiko sistema, urediti komunikacijo med odjemalcem in podatkovno shrambo ter poskrbeti, da je bil sloj dovolj zmogljiv za obdelavo zahtevkov vseh uporabnikov.

Za tehnološko osnovo smo uporabili programski jezik PHP (glej poglavje 3.5) in ogrodje Yii (glej poglavje 3.6). Odločitev za programski jezik PHP in Yii je izvirala predvsem iz predhodnih lastnih projektov in dobrega poznavanja principov dela z njimi.

Na podlagi funkcionalnih zahtev smo razvili podatkovni model (priloga A) in definirali relacije med posameznimi entitetami, ki jih bomo potrebovali v sistemu. S pripravljenim podatkovnim modelom smo definirali podatkovno shemo v podatkovni bazi in povezali Yii na izbrano podatkovno shrambo. Z orodjem Yii smo iz pripravljene podatkovne baze ustvarili osnovne datoteke z izvorno kodo (CRUD-sistem, opisan v poglavju 3.6)). Sistem CRUD je že pripravil vse osnovne poglede za urejanje in pregledovanje podatkov v podatkovni shrambi.

S pomočjo definiranih tabel v podatkovni bazi so se izgradili tudi modeli oz. programski objekti, s katerimi je mogoče programsko dostopati do zapisov v podatkovni shrambi, ne da bi bilo zato treba pisati kakršne koli poizvedbe (SQL) na podatkovno shrambo. Programski dostop do podatkov v podatkovni shrambi imenujemo ORM (*object relational mapping*) oz. po slovensko objektno-relacijska preslikava. ORM je tehnika programiranja, ki omogoča preslikavo programskih objektov v strukturo, ki jo je mogoče shraniti v podatkovno shrambo. Yii ORM je zelo dobro dodelan in poleg osnovnega shranjevanja v podatkovno shrambo podpira tudi relacije in tuje ključe, kar v praksi pomeni, da jo, če ima nek objekt neko lastnost, ki je zapisana v drugem objektu, zna ORM-sistem ob zahtevi lastnosti avtomatsko naložiti iz podatkovne shrambe. Tak način delovanja ogrodja Yii nam je omogočil enostaven in relativno hiter razvoj sistema.

Za programiranje pravil in načinov prikaza podatkov na uporabnikovem brskalniku smo uporabili vgrajen način za komunikacijo z odjemalcem, to je princip MVC (več o tem v poglavju 3.6). Tekmovalni del platforme je zelo pomemben, zato smo na tem delu želeli optimizirati komunikacijo med odjemalcem in aplikacijskim slojem. V sklopu optimiziranja komunikacije smo implementirali lastne REST-klice, ki omogočajo čim manjšo in problemsko

orientirano komunikacijo.

Določene poizvedbe v podatkovno shrambo in deli programske kode se skozi izvajanje aplikacijskega sloja ponavljajo, zato smo uporabili vgrajeno predpomnjenje v Yii-ogrodju in s tem pohitrili obdelavo zahtev uporabnikov na strežnik. Predpomnjenje pomeni, da rezultate poizvedb v podatkovno shrambo oz. rezultate izvajanja programske kode shranimo v posebno podatkovno shrambo, za katero velja, da je dostop do nje hitrejši kot ponovno izvajanje bloka programske kode.

V našem primeru smo za posebno podatkovno shrambo za predpomnjenje uporabili storitev Memcache. Memcache je podatkovna shramba tipa ključ-vrednost, ki za hranjenje podatkov uporablja pomnilnik strežnika. Memcache nima obstojnega shranjevanja podatkov, kar pomeni, da se mora koda aplikacijskega sloja ves čas zavedati, da morda podatka iz predpomnilnika ne bo dobila in da bo morala del programske kode izvesti ponovno. Podatkovna shramba Memcache ima vgrajene tudi časovnike, ki omogočajo, da lahko programska koda aplikacijskega sloja nastavi, kako dolgo naj se nek podatek hrani v predpomnilniku. Po pretečenem času veljavnosti zapisa v podatkovni shrambi se podatek avtomatsko izbriše.

Ko pa gledamo sistem kot celoto, je aplikacijski sloj prvi, ki se nahaja na naši infrastrukturi in mora biti dovolj zmogljiv, da lahko postreže vsem zahtevam uporabnikov. Na tem sloju smo morali poskrbeti, da lahko sloj deluje hkrati na več strežnikih, s čimer dosežemo večjo zmogljivost in toleranco na izpade posameznih strežnikov. Porazdelitev sloja na več strežnikov smo dosegli tako, da smo v DNS-zapis ene domene dodali več zapisov DNS A za vsak naš aplikacijski strežnik posebej.

DNS strežnik ves čas spreminja vrstni red zapisov DNS A za domeno, tako da vsak uporabnik, ki želi dostopati do neke domene, dobi različen vrstni red DNS-zapisov (razvidno v dveh različnih testih v primeru 4.1 v razdelku odgovora `ANSWER SECTION`). Brskalnik se glede na vrstni red DNS-zapisov začne povezovati na strežnik, najprej na prvega, nato na naslednjega itd.

```
TEST 1:
dig a bober.acm.si

; <<>> DiG 9.9.3-P2 <<>> a bober.acm.si
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8578
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2,
   ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;bober.acm.si.                  IN      A

;; ANSWER SECTION:
bober.acm.si.           86400   IN      A       193.2.76.43
bober.acm.si.           86400   IN      A       193.2.76.42
bober.acm.si.           86400   IN      A       193.2.76.37

;; AUTHORITY SECTION:
acm.si.                 7200    IN      NS      dns1.acm.si.
acm.si.                 7200    IN      NS      dns2.acm.si.

;; Query time: 22 msec
;; SERVER: 193.9.21.12#53(193.9.21.12)
;; WHEN: Sat Mar 01 19:48:23 CET 2014
;; MSG SIZE  rcvd: 111

TEST 2:
dig a bober.acm.si

; <<>> DiG 9.9.3-P2 <<>> a bober.acm.si
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 8578
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 2,
```

```
    ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;bober.acm.si.                  IN      A

;; ANSWER SECTION:
bober.acm.si.           86400   IN      A       193.2.76.37
bober.acm.si.           86400   IN      A       193.2.76.43
bober.acm.si.           86400   IN      A       193.2.76.42


;; AUTHORITY SECTION:
acm.si.                 7200    IN      NS      dns1.acm.si.
acm.si.                 7200    IN      NS      dns2.acm.si.

;; Query time: 22 msec
;; SERVER: 193.9.21.12#53(193.9.21.12)
;; WHEN: Sat Mar 01 19:49:12 CET 2014
;; MSG SIZE  rcvd: 111
```

Primer 4.1: DNS-poizvedbi za naslov bober.acm.si

V primeru, da eden izmed strežnikov ni dostopen (to pomeni, da na spletnih vratih 80, 443 ni dosegljiva nobena storitev), imajo današnji brskalniki (Internet Explorer $\geq$ 8, Opera, Safari $\geq$ 4.0.4, Mozilla Firefox $\geq$ 3.4, Google Chrome $\geq$ 4.0 [15]) vgrajen sistem v sodelovanju z lokalnim DNS-razreševalnikom, da poskusijo vzpostaviti povezavo z naslednjim IP-naslovom, ki ga vrne DNS-strežnik, in tako poskusijo priti do aktivnega strežnika. V primeru izpada posameznih fizičnih strežnikov (nedostopnost spletnih vrat) smo z več DNS-zapisi in več strežniki dosegli avtomatsko preusmeritev uporabnikov na delujoče strežnike. Pomanjkljivost takega načina delovanja je, da določeni starejši brskalniki še nimajo vgrajene podpore za ta način delovanja in da v tem primeru preusmeritve uporabnikov ne delujejo, kar pomeni, da lahko pride do situacije, ko na nekem strežniku storitev ne de-

luje pravilno in pride do napake v komunikaciji s podatkovno shrambo, ampak ti brskalniki ne bodo poskusili dostopati do drugega strežnika. Za boljšo rešitev preusmeritve uporabnikov bi morali uporabiti dodatne vmesne (*proxy*) strežnike, ki bi hkrati znali preverjati pravilnost delovanja storitev aplikacijskega sloja, oz. uporabiti dodatno knjižnico za DNS-strežnik, ki bi omogočila avtomatsko preverjanje delovanja strežnikov in izločitev nedelujočih IP-naslovov strežnikov iz seznama DNS-zapisov.

Ko brskalniki razrešijo domeno bober.acm.si, se preko IP-naslova neposredno povežejo z aplikacijskim strežnikom. Za aplikacijski strežnik smo uporabili napreden odprtokodni strežnik Nginx, ki je v osnovi neke vrste vmesni (*proxy*) strežnik. Nginx smo nastavili, da dostope do statičnih datotek (CSS, JavaScript, slike) lokalno predpomni v pomnilniku in jih neposredno obdela, ostale zahtevke po dinamični vsebini pa posreduje naprej v obdelavo PHP-aplikacijskemu strežniku. Za PHP-aplikacijski strežnik smo uporabili posebno različico PHP-strežnika PHP-FPM (*FastCGI Process Manager*) [26, 57].

Ker smo porazdelili izvajanje aplikacijskega sloja na več strežnikov, so se pojavile težave s stanjem uporabnikov. Ker je HTTP-protokol brez stanj, smo uporabili sistem sej (sejni piškotki), da lahko sledimo uporabniku skozi njegove zahteve na strežnik. Podatki seje uporabnikov se običajno hranijo lokalno na posameznem strežniku. Pri porazdeljeni postavitvi aplikacijskega sloja pa podatkov o sejah ni mogoče več hraniti lokalno na posameznem strežniku, ampak je treba za hranjenje podatkov uporabiti neko skupno podatkovno shrambo. Če bi pri porazdeljenem sistemu uporabili lokalno hranjenje, bi se nam lahko zgodilo, da bi uporabnik dostopal do enega strežnika in dobil podatke za vzpostavitev seje, nato bi naredil nov zahtevek in potencialno bi bilo mogoče, da bi bil drugi zahtevek preusmerjen na drug aplikacijski strežnik, ki ne bi poznal seje, s katero bi se predstavil uporabnik, in sistem ne bi deloval. Za hranjenje podatkov o sejah za uporabnike v našem sistemu smo uporabili skupno podatkovno shrambo MySQL Cluster.

4.3.3 Podatkovna shramba

Jedro večine današnjih aplikacij so podatki in težava, ki se pojavi, je, kako podatke porazdeliti med več strežnikov in zagotoviti hiter in zanesljiv dostop. Do enakega zaključka smo prišli tudi pri tekmovalnem sistemu, zato je bilo opravljenega veliko raziskovanja in testiranja, da smo prišli do rešitve, ki bo zadoščala potrebam celotnega sistema. Po pregledu zahtev sistema in pregledu odprtokodnih rešitev podatkovnih baz smo se odločili, da uporabimo gručno različico podatkovne baze MySQL Cluster (opisano v poglavju 3.7), za katero smo našli boljšo dokumentacijo in pripravljene primeri uporabe na spletu [35, 40].

Eden ključnih razlogov, zakaj smo se dejansko odločili za podatkovni sistem MySQL Cluster, je dejstvo, da so razvijalci MySQL Cluster v letu 2013 z verzijo 7.3 dodali podporo za tuje ključe in omejitve v podatkovni pogon NDB, kar pomeni, da so s tem pokrili enake funkcionalnosti, kot jih ima InnoDB s pogonom NDBCLUSTER, le da v NDBCLUSTER deluje v gručni različici MySQL in omogoča, da imamo podatke porazdeljene in sinhronizirane na več različnih strežnikih. Porazdelitev na več strežnikov nam prinese decentralizacijo sistema, kar pomeni, da nimamo ene same točke v sistemu, od katere bi bil odvisno delovanje celotnega sistema. Poleg tega nam tak sistem z večanjem števila strežnikov in strojnih virov omogoča povečevanje kapacitet in zmogljivosti, kar je pomembno za rabo tekmovalne platforme v prihodnosti, saj pričakujemo večje število tekmovalcev in uporabo platforme tudi v državah izven meja Slovenije. MySQL Cluster nam je s svojo arhitekturo omogočil zanesljivo podatkovno shrambo, v katero se zapisujejo vsi potrebni podatki za uspešno delovanje celotne tekmovalne platforme.

Slaba lastnost MySQL Cluster je licenca, izdan je namreč pod licenco GPLv2, kar pomeni, da podatkovnega sistema ni mogoče uporabiti v komercialne namene in zapakirati rešitve kot celotnega produkta/paketa. Je pa mogoče brez kršitev uporabiti MySQL Cluster v komercialnih projektih, če se namestitev gruče loči od namestitve komercialnega projekta, običajno z navodili za namestitev podatkovnega sistema s strežnikov MySQL-razvijalcev.

Odlična stvar MySQL-sistemov je, da za komercialne namene omogočajo tudi možnost nakupa licence, zakupa podpore in pomoči, kar omogoča lažje, hitrejše in bolj efektivno delo.

V okviru diplomskega dela in postavitve sistema MySQL Cluster za potrebe izvedbe tekmovanja Bober v Sloveniji in Srbiji v letu 2013 so nastala tudi navodila za postavitev MySQL Cluster v praksi [20].

Podatkovni model Bober

Podatkovni model tekmovalnega sistema Bober je sestavljen iz 35 relacij oz. tabel, shema relacij in povezav med relacijami se nahaja v prilogi A. V relaciji `users` (uporabniki) se nahajajo osnovi podatki o uporabnikih (uporabniško ime, geslo, e-poštni naslov, čas dodajanja uporabnika, čas zadnje prijave uporabnika, ali je uporabnik superadministrator ali ne, ter še nekaj ostalih prehodnih atributov relacije, ki so potrebni za delovanje registracijskega postopka). Za primere potreb po razširljivosti profila uporabnikov sta dodani relaciji `profiles` in `profiles_fields`, preko katerih je mogoče dinamično, brez dodatnega programiranja dodati nova polja v profilu posameznega uporabnika. Seje uporabnikov sistema so zaradi uporabe porazdeljenih sistemov zapisane v posebni relaciji `YiiSession`, ki je sestavljena iz treh enostavnih atributov, in sicer identifikatorja seje oz. piškotka v brskalniku (`id`), podatka o tem kako dolgo je veljavna seja (`expire`) in podatkov, shranjenih v seji (`data`). Za potrebe boljše kategorizacije tekmovanj, šol in uporabnikov je uporabljenih nekaj relacij za šifrante, in sicer relacija `school` (šole), `school_category` (kategorije šol, osnovna šola, srednja šola itd.), `region` (regije, goriška, savinjska ...), `municipality` (občine), `language` (jeziki), `country` (države), `country_language` (jeziki v državi). Šifranti se uporabljajo v ostalih relacijah in so med seboj povezani s tujim ključem. V relaciji `school_mentor` (mentorji na šoli) se nahaja informacija o tem, kateri uporabnik v sistemu je mentor na kateri šoli, vsak uporabnik je lahko mentor na več šolah, prav tako ima lahko vsaka šola več mentorjev. V relaciji `country_administrator` (državni administratorji) se nahaja informacija, kateri uporabniki so administratorji

v posameznih državah.

Naloge oz. vprašanja so v sistem vnešena preko relacije `question` (vprašanja). Relacija `question` ima atribute `id`, `country_id` (povezava s tujim ključem na državo), `type` (vrsta naloge, ali gre za navadno nalogo ali interaktivno nalogo, trenutno so podprte le interaktivne naloge), `title` (naziv naloge), `text` (besedilo naloge, če gre za navadno nalogo), `data` (dodatni podatki o nalogi), `version` (verzija naloge, pomembno pri uvažanju interaktivnih nalog), `verification_function_type` (vrsta načina preverjanja pravilnost naloge, interni način, kjer sistem Bober dejansko ve, kateri odgovor je pravilen, JavaScript-način, kjer se glede pravilnosti odgovora povpraša evalvacijsko funkcijo interaktivne naloge), `verification_function` (funkcija za preverjanje pravilnosti odgovora za interaktivne naloge, v to polje se lahko zapiše statični odgovor na interaktivno nalogo ali pa JavaScript-evalvacijska funkcija), `authors` (avtorji naloge), `css` (dodatni CSS-stili, ki jih je mogoče vrniti v interaktivno nalogo za boljšo vizualizacijo naloge v tekmovalnem vmesniku Bober).

Osnovna relacija `question` se povezuje z ostalimi relacijami, ki dopolnjujejo naloge z dodatnimi informacijami. V relaciji `question_answer` (odgovori na vprašanja) se za vsako nalogo tipa navadna naloga nahajajo odgovori na vprašanja. V relaciji `question_translation` (prevod vprašanja) in `question_answer_translation` (prevodi odgovorov na vprašanja) se zapišejo prevodi nalog in odgovorov na naloge v druge jezike, kar pride v upoštev za države, ki imajo več uradnih jezikov. Da so lahko viri nalog, kot so slike, besedila, skripte, zaščiteni pred nepooblaščenim vpogledom, so vse datoteke zapisane v relaciji `question_resource` (viri vprašanj). Relacija je sestavljena iz atributov `id`, `question_id` (povezava s tujim ključem z vprašanjem), `language_id` (povezava s tujim ključem z jezikom), `type` (vrsta vira: naloga, razlaga rešitve ali evalvacijska funkcija), `path` (pot do datoteke, virtualni imenik, to je pomembno za uvožene interaktivne naloge, ki imajo znotraj statične HTML vsebine relativno pot do ostalih virov, recimo slika1.jpg ni v istem imeniku kot HTML-naloge, ampak

v mapi „data“), `filename` (ime datoteke vira), `file_type` (vrsta datoteke, pomembna informacija, da lahko programsko pravilno vračamo vsebino brskalniku, da se brskalnik ne zaveda virtualnosti datotek), `data` (binarni podatki virov), `start_up` (vsaka interaktivna naloga ima eno začetno datoteko, ponavadi je to index.htm, HTML-datoteka z nalogo).

Tekmovanje se v podatkovnem modelu zapiše v relaciji `competition` (tekmovanje). V relaciji tekmovanje se določi `name` (ime tekmovanja), `active` (ali je tekmovanje že aktivno ali ne), `timestamp_start` (datum in čas pričetka tekmovanja), `timestamp_stop` (datum in čas zaključka tekmovanja), `type` (vrsta tekmovanja: šolsko, državno), `public_access` (ali je tekmovanje že javno dostopno za vadbo), `duration` (čas tekmovanja v minutah), `timestamp_mentor_results` (datum in čas, od katerega dalje mentorji vidijo rezultate tekmovanja), `timestamp_mentor_awards` (datum in čas, od katerega dalje mentorji vidijo, kakšne nagrade so dobili njihovi učenci), `timestamp_mentor_advancing_to_next_level` (datum in čas, od katerega dalje mentorji vidijo, kdo bo napredoval na naslednjo stopnjo tekmovanja). Ime tekmovanja je mogoče prevesti tudi v druge jezike z zapisom podatka v relacijo `competition_translation` (prevod tekmovanja). Vsakemu tekmovanju je mogoče določiti, v kateri državi se bo tekmovanje izvajalo, to se določi preko relacije `competition_country` (država tekmovanja). V relaciji `competition_category` (kategorija tekmovanja) se nahaja seznam vseh kategorij tekmovanja (Bobrček, Mladi bober ...) in vseh omejitev, v katerem razredu in stopnji izobraževanja mora biti tekmovalec, ki pristopi k posameznemu tekmovanju. Prevodi za imena kategorij tekmovanja se nahajajo v relaciji `competition_category_translation` (prevodi imen kategorij tekmovanja). V relaciji `competition_category_active` (aktivne kategorije tekmovanja) se določijo nastavitve tekmovalne kategorije specifično za neko tekmovanje, določi se število vprašanj, število potrebnih točk za posamezno priznanje ter število tekmovalcev, ki bo napredovalo na naslednjo stopnjo tekmovanja. V relaciji `competition_committee` (komisija tekmovanja) se določi, kateri uporabniki v sistemu so člani ko-

misije posameznega tekmovanja. V relaciji `competition_question_difficulty` (težavnostne kategorije vprašanj na tekmovanju) se nahaja šifrant težavnostih stopenj vprašanj na tekmovanju in informacija o številu točk, ki jih tekmovalec dobi v primeru pravilnega oz. izgubi v primeru napačnega odgovora. Imena težavnostih stopenj vprašanj je mogoče tudi prevesti v druge jezike z zapisom podatka v relacijo `competition_question_difficulty_translation`.

Pri sestavljanju tekmovanja je treba določiti vprašanja na tekmovanju za vsako kategorijo tekmovalcev posebej, to informacijo se zapiše v relacijo `competition_question`, kjer se zapiše podatek, katera vprašanja so na posameznem tekmovanju, in `competition_question_category`, preko katere se določi, katero vprašanje na tekmovanju bo za tekmovalce posamezne kategorije ter s kakšno težavnostno stopnjo bo vprašanje vrednoteno.

Na vsako tekmovanje se morajo šole prijaviti, podatek o tem se zapiše v relacijo `competition_category_school`, pri tem se tudi določi, v katerih kategorijah tekmovanja bodo učenci s te šole tekmovali. Poleg tega se morajo na tekmovanja prijaviti tudi mentorji na šolah, kar se zapiše v relaciji `competition_category_school_mentor`. V tej relaciji se zapiše dostopna koda do tekmovanja, ki jo mentorji prejmejo in posredujejo na dan tekmovanja svojim učencem. Preko uporabljene dostopne kode nato tekmovalni sistem tekmovalca poveže s pravilnim mentorjem, kategorijo tekmovanja, šolo in državo. Podatki o tekmovalcu se zapišejo v relaciji `competition_user`. Poleg prej naštetega se v tej relaciji shrani tudi `first_name` (ime), `last_name` (priimek), `gender` (spol), `class` (razred), informacije o diskvalifikaciji, `advancing_to_next_level` (ali tekmovalec napreduje na naslednjo stopnjo tekmovanja), `award` (ali je tekmovalec prejel kakšno nagrado, `start_time` (datum in čas pričetka tekmovanja), `finish_time` (datum in čas zaključka tekmovanja), `finished` (ali je tekmovalec zaključil tekmovanje), informacija o številu doseženih točk, `ip_start` (IP-naslov tekmovalca ob začetku tekmovanja), `ip_stop` (IP-naslov tekmovalca ob zaključku tekmovanja). Ko se tekmovalcu pripravi

tekmovanje, se podatek o vrstnem redu vprašanj in naključnem številu za interaktivne naloge zapiše v relacijo `competition_user_questions`. Če so tekmovalna vprašanja navadna vprašanja (ne interaktivne naloge) v sistemu, se poleg vrstnega reda vprašanj zapiše tudi vrstni red odgovorov, in sicer v relaciji `competition_user_question_answer`. Ko tekmovalec rešuje naloge, se shranjujejo njegovi odgovori v relacijo `competition_user_question`, in sicer na sledeč način: če gre za navadna vprašanja, se zapiše podatek o izbranem odgovoru v atribut `question_answer_id` (povezan tuj ključ na odgovor na vprašanje), če pa gre za interaktivno nalogo, se izbrani odgovor na vprašanje zapiše v atribut `custom_answer` (poljuben odgovor, odgovor, ki ga zna sprejeti evalvacijska funkcija interaktivne naloge in na podlagi tega sporočiti, ali je odgovor na vprašanje pravilen ali ne).

Poglavje 5

Testiranje in ovrednotenje

5.1 Opis sistema

Tekmovalni sistem Bober je bil postavljen v tri virtualne strežnike, ki so delovali na dveh fizičnih strežnikih.

Strojna oprema gostiteljskih strežnikov je taka:

- CPE: 32x jedro AMD Opteron(TM) Processor 6272, 1,4 Ghz
- Pomnilnik: 128 GB
- Disk: 5,5 TB

Programska oprema virtualnih strežnikov:

- Operacijski sistem: Linux Ubuntu 12.04.3 LTS
- Spletni strežnik Nginx 1.1.19
- PHP 5.3.10 s FPM in CLI-modulom
- MySQL Cluster GPL 7.3.2
- MySQL Proxy 0.8.1

- Memcached 1.4.13
- Zabbix Agent v 2.0.10
- ntpdate 4.2.6p3

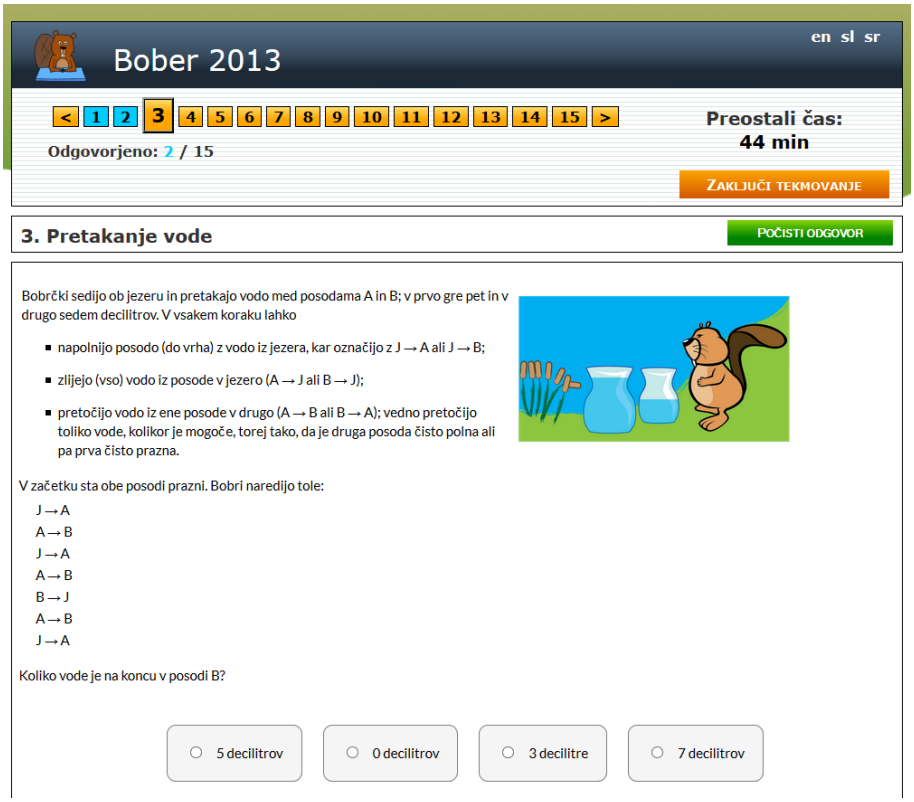
5.2 Funkcionalnosti

Glede na zahteve sistema smo podprli večino funkcionalnosti, to dejstvo podpirajo zaslonske slike aplikacije in delujoči sistem na spletnem naslovu: `https://bober.acm.si`

5.2.1 Tekmovanje

Slika 5.1: Vhodna stran tekmovalnega sistema, preko katere lahko tekmovalci vpišejo svoje podatke in kodo za dostop in pričnejo s tekmovanjem.

Na vhodni strani na sliki 5.1 tekmovalec vpiše svoje osebne podatke in vnese kodo za dostop. Če so izpolnjeni podatki pravilni, uporabnika preusmeri na tekmovalni vmesnik.



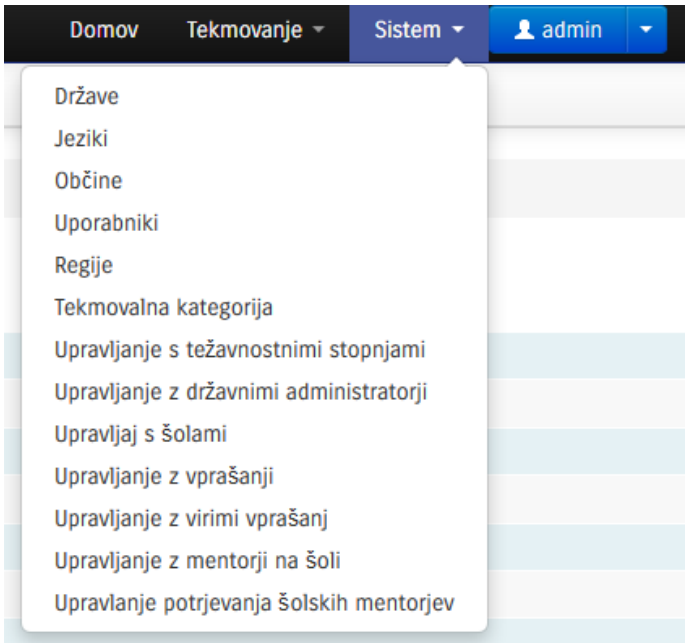
Slika 5.2: Videz tekmovalnega vmesnika.

V zgornji navigacijski vrstici tekmovalnega vmesnika na sliki 5.2 je mogoče preklapljati med nalogami, svetlomodro obarvana naloga pomeni, da je tekmovalec na vprašanje že odgovoril. Pod navigacijsko vrstico je informacija o številu odgovorjenih nalog v odvisnosti od števila vseh nalog. Zgoraj desno je mogoče preklopiti tekmovalni vmesnik v druge jezike, trenutno podprti jeziki so angleščina, slovenščina in srbščina. Na desni, vzporedno z navigacijsko vrstico, je prikazan preostali čas tekmovanja, ki ga ima tekmovalec na voljo za reševanje nalog. Pod preostalim časom se nahaja gumb za zaključek tekmovanja. Ob kliku na ta gumb se prikaže opozorilo, ki tekmovalca opozori, na koliko vprašanj ni odgovoril in da po zaključku tekmovanja ne bo več

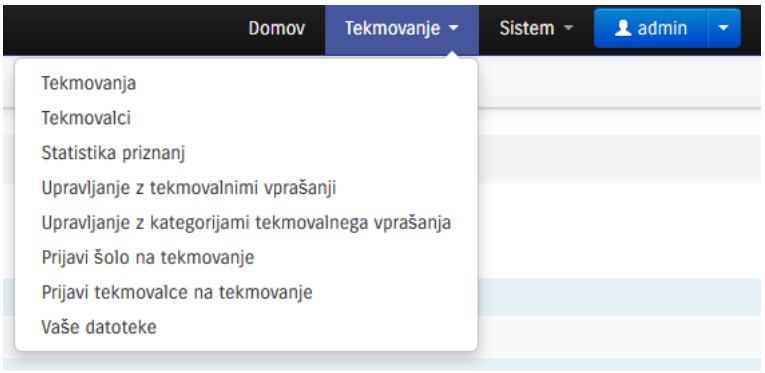
mogoče spreminjati odgovorov na naloge. Pod zgornjim funkcijskim oknom se nahaja okvir z nazivom naloge in gumbom za izbris odgovora na nalogo, s katerim lahko tekmovalci izbrišejo odgovor na nalogo, saj napačen odgovor na vprašanje tekmovalcu prinaša negativne točke. Pod tem okvirjem pa sledijo vsebina naloge in odgovori.

5.2.2 Upravljanje

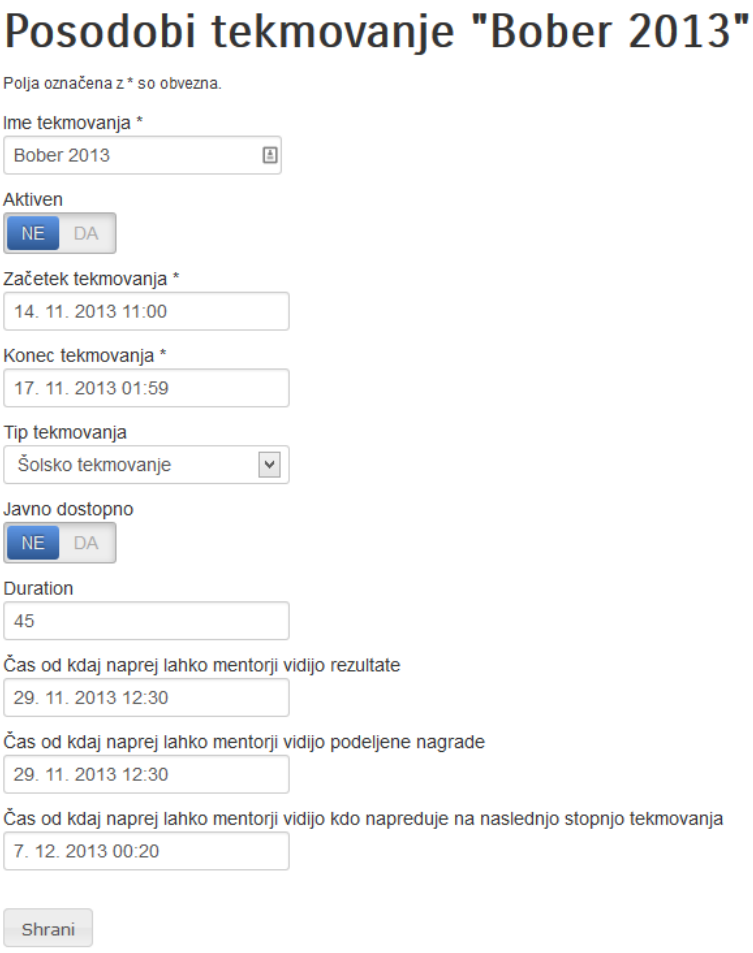
Z vmesnikom za upravljanje na slikah 5.3 in 5.4 je mogoče nastavljati vse potrebne podatke za izvedbo tekmovanja. Spremeniti je možno vse podatke, države, jezike, tekmovalne kategorije, vprašanja, tekmovanja itd. S pogledom za urejanje tekmovanja na sliki 5.5 je mogoče urejati vse podatke o tekmovanju, ime tekmovanja, čas začetka in konca tekmovanja, tip tekmovanja, čas tekmovanja in kdaj posamezne informacije o tekmovanju vidijo mentorji tekmovalcev. V administraciji so na podoben način, kot na sliki 5.5, implementirani tudi vsi ostali obrazci za urejanje.



Slika 5.3: Pregled administratorskih funkcionalnosti za administratorje.

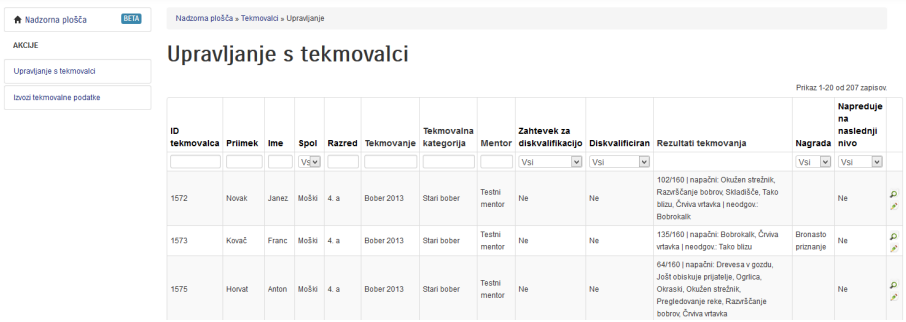


Slika 5.4: Pregled administratorskih funkcionalnosti glede tekmovanja.



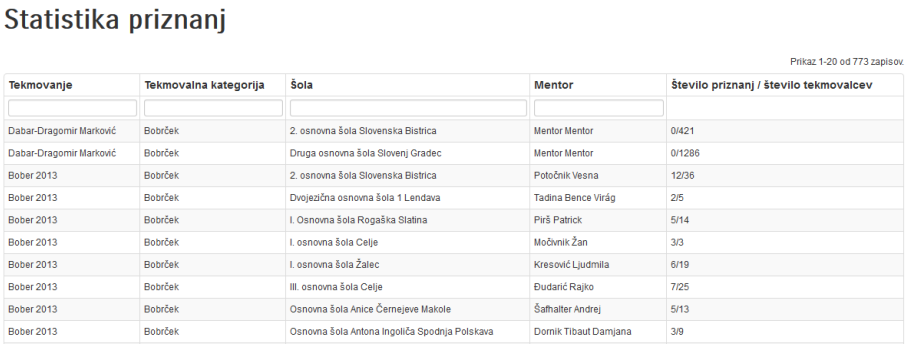
Slika 5.5: Administratorski pogled za urejanje tekmovanja.

5.2.3 Mentorstvo



Slika 5.6: Pogled mentorja, ki upravlja s svojimi tekmovalci.

Vsak mentor se lahko prijavi v sistem in pregleduje svoje tekmovalce (pogled na sliki 5.6), mentor vidi točke in naloge, ki so jih tekmovalci narobe rešili, dobljena priznanja in status napredovanja na naslednji nivo tekmovanja. Dolžnost mentorjev je, da uredijo podatke o svojih tekmovalcih, torej da jim popravijo napake v imenih, priimkih in razredih, tako da so podatki urejeni pred tiskom priznanj.



Slika 5.7: Statistika priznanj.

Za koordinatorje šol in administratorje smo pripravili statistiko priznanj (pogled na sliki 5.7), preko katere je mogoče analizirati, koliko priznanj je dobila posamezna šola v posamezni kategoriji tekmovanja.

Z izdelanim sistemom je bilo v Sloveniji novembra 2013 uspešno izvedeno tekmovanje na nivoju šol, ko se je udeležilo 12.331 tekmovalcev, in januarja

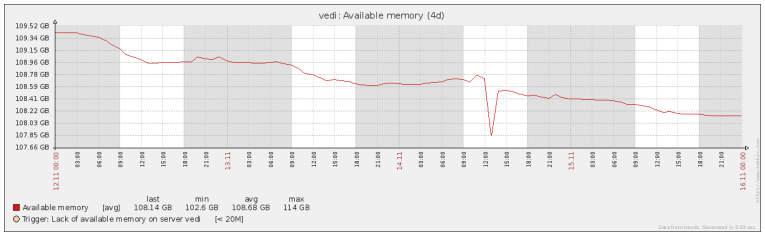
2014 državno tekmovanje, ko se je udeležilo 320 tekmovalcev. Ob razvoju tekmovalnega sistema so nas povabili v Srbijo, kjer smo jim predstavili idejo tekmovanja in že decembra 2013 so z našimi sistemom izvedli tekmovanje na nivoju šol, kjer smo za prvo leto dosegli visoko število tekmovalcev (6.272). Kvalitativno je bil torej sistem večkrat preizkušen in je prestal vse zahteve programskih svetov, ki so organizirali in izvedli tekmovanje, zato si upamo trditi, da smo implementirali potrebne funkcionalne zahteve sistema (poglavje 1.4).

Nedokončanih je ostalo nekaj funkcionalnosti, in sicer ni stroge delitve med državnimi administratorji in člani tekmovalne komisije, prav tako ni implementirana neposredna API-komunikacija s skladiščem nalog, tako da je treba naloge izvoziti iz sistema za pripravljanje nalog in ga uvoziti v tekmovalno platformo. Ideja o generiranju tekmovalnih priznanj in priznanj za učitelje je bila preprosto prezahtevna za implementacijo znotraj tekmovalne platforme, zato smo uporabili zunanjo aplikacijo za pripravo PDF-datotek, ki je bila implementirana že v preteklih letih [43]. Nedokončan je ostal način za vadbo, kjer bi lahko tekmovalci vadili tekmovanja s preteklimi tekmovalnimi nalogami in bi se jim ob koncu tekmovanja prikazali pravilni odgovori in obrazložitve nalog. Poleg tega pa je tako v tekmovalnem kot v administrativnem delu ostalo veliko možnosti za izboljšavo uporabniške izkušnje pri uporabi sistema. Realno bi sistem potreboval še nekaj mesecev razvoja, da bi se dokončale vse funkcionalnosti.

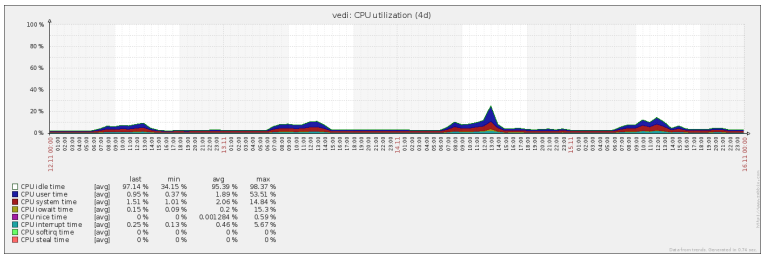
5.3 Kvantitativno ovrednotenje

Za meritve stabilnosti sistema med tekmovanjem smo uporabili več meritvenih sistemov, saj smo uporabili strežnik Zabbix [56], ki je spremljal vse pomembne sistemske parametre: uporabljeni pomnilnik (glej sliko 5.8), uporaba CPE (glej sliko 5.9), število zahtevkov na strežnik (glej sliko 5.10), dostopnost vseh strežnikov v gruči celotnega tekmovalnega sistema itd. To je bilo notranje spremljanje stanja strežnikov, za zunanje spremljanje strežnika

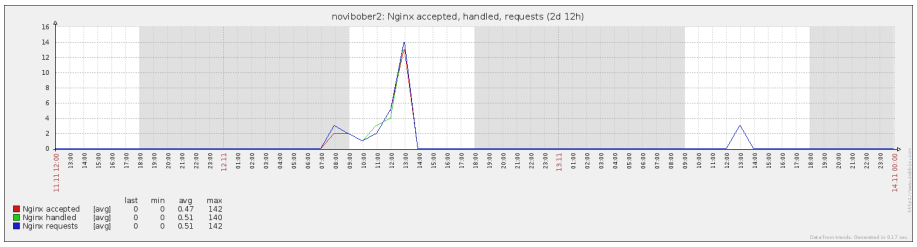
smo uporabili komercialno storitev Pingdom [44], ki spremlja dosegljivost posameznih strežnikov na nivoju HTTP/HTTPS-protokola (dostop do aplikacijskega strežnika), meri odzivni čas (ping) do strežnika in v primeru izpada takoj obvesti administratorje preko SMS-sporočila in e-pošte. Tekom vseh tekmovanj so bili vsi strežniki skoraj ves čas stabilni, razen manjšega izpada programja podatkovne baze zaradi preobremenjenosti gostiteljska strežnika z obdelavami drugega virtualnega strežnika.



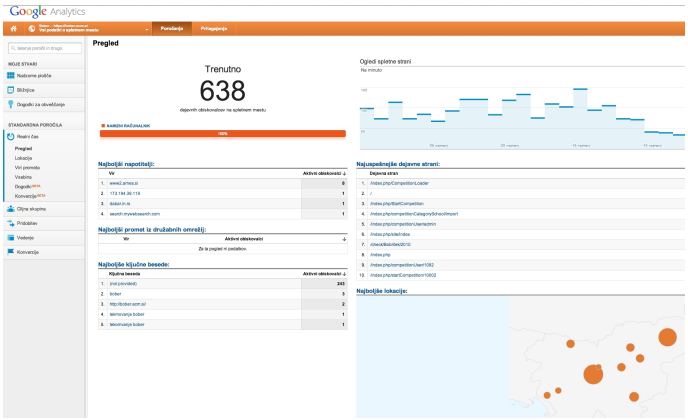
Slika 5.8: Poraba pomnilnika med izvajanjem šolskega tekmovanja Bober v Sloveniji v letu 2013.



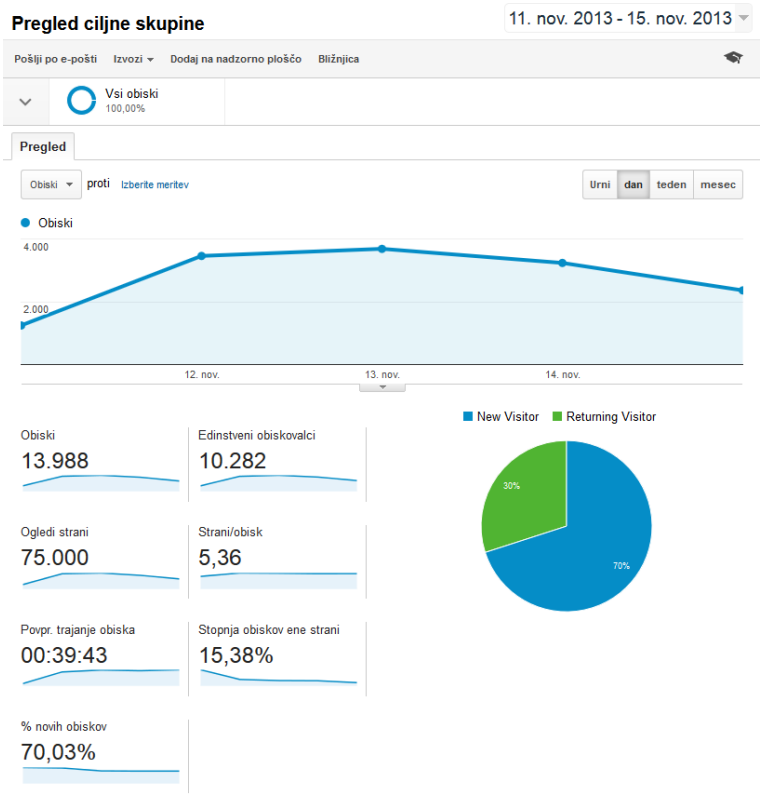
Slika 5.9: Poraba CPE med izvajanjem šolskega tekmovanja Bober v Sloveniji v letu 2013.



Slika 5.10: Pregled števila zahtevkov na sekundo na strežnik med izvajanjem šolskega tekmovanja Bober v Sloveniji v letu 2013.



Slika 5.11: Realno-časovna Googlova analitika, iz katere je razvidno hkratno število uporabnikov med izvajanjem šolskega tekmovanja Bober v Sloveniji v letu 2013.



Slika 5.12: Pregled števila obiska spletne strani bober.acm.si v tednu tekmovanja Bober v Sloveniji 2013.

Za potrebe analiziranja skupne zmogljivosti celotne postavitve tekmovalnega sistema smo uporabili Google Analytics [17]. S pomočjo Google Analytics smo spremljali število hkratnih uporabnikov sistema (glej sliko 5.11). Poleg spremljanja skupnega števila uporabnikov smo beležili tudi vse dogodke v brskalniku (kliki na gumbe, premikanje med nalogami itd.), ki se dogajajo med tekmovanjem uporabnikov. S pomočjo analize skupnega števila obiskov na sliki 5.12 smo ugotovili, da smo v času tekmovanja Bober v Sloveniji v letu 2013 zabeležili 13.988 obiskovalcev in 75.000 ogledov spletnih strani ter da je uporabnik na strani povprečno preživel skoraj 40 minut, kar se sklada s 45-minutno časovno omejitvijo tekmovanja.

5.4 Povzetek

Uporabniški vmesnik in aplikacijski sloj sta bila popolnoma stabilna in ni bilo odkritih nobenih napak. Na sloju podatkovne shrambe pa smo med šolskim tekmovanjem v Sloveniji leta 2013 zabeležili 10-minutni popolni izpad delovanja, a smo po analizi izrednega dogodka ugotovili, da je nekdo med tekmovanjem na istem fizičnem strežniku sprožil obdelavo, ki je zasedla celoten pomnilnik strežnika, virtualni strežniki, v katerih so bile zagnane storitve Bobra, pa niso imeli zagotovljenega pomnilnika, kar je sprožilo, da je jedro gostitelja ugasnilo procese MySQL Cluster, ki so na sistemu vzeli velik del pomnilnika, zaradi česar se je celotna gruča podatkovnih vozlišč na strežniku ugasnila. V času nedelovanja podatkovne shrambe se odgovori tekmovalcev na vprašanja niso shranjevali. Nadzorna vozlišča MySQL Cluster so takoj po zaznani napaki avtomatsko zagnala nova podatkovna vozlišča. Podatkovna shramba se je po nepričakovani zaustavitvi uspešno ponovno vzpostavila, in to samodejno, brez posredovanja administratorjev. Na podlagi tega sklepamo, da je delovanje sloja podatkovne shrambe uspešno. V uporabi smo imeli tri virtualne strežnike, za uspešno delovanje sta potrebna vsaj dva od treh. Do izpada je prišlo na gostiteljskem strežniku, kjer sta bila prisotna dva virtualna strežnika Bober, kar je povzročilo, da je ostal prisoten samo še

en virtualni strežnik na drugem gostiteljskem strežniku, kar pa ni dovolj in prišlo je do izpada porazdeljene podatkovne baze.

V prihodnje bi bilo treba zagotoviti, da je vsak virtualni strežnik Bober nameščen na ločeni strojni opremi oz. da virtualizacijska tehnologija zagotavlja vire za vsak virtualni strežnik posebej, tako da ena nepredvidena situacija na gostiteljskem strežniku ne more vplivati na delovanje več kot enega virtualnega strežnika. Glede na izkušnje iz prvega šolskega tekmovanja smo pri naslednjih tekmovanjih opozorili vse uporabnike, ki si delijo gostiteljski strežnik, naj v času tekmovanja ne uporabljajo strežnika, in tako smo izvedli šolsko tekmovanje decembra 2013 v Srbiji in državno tekmovanje januarja 2014 v Sloveniji brez napak.

Za namene dodatne zaščite pred izpadi aplikacijskega sloja in podatkovne shrambe med tekmovanjem bi predlagali, da bi lahko naredili dnevniško zapisovanje odgovorov na nek tretji strežnik, kjer bi se podatki pošiljali preko GET-parametrov HTTP-zahtevkov, tako bi imeli dodaten zapis vseh odgovorov tekmovalcev, da bi v primerih popolnega izpada podatkovne shrambe imeli možnost rekonstrukcije odgovorov.

Druga varianta varovanja podatkov tekmovalca bi bila asinhrona komunikacija s strežnikom, ko bi se torej ves čas tekmovanja vodil dnevnik dogodkov oz. odgovorov na vprašanja, na strežnik bi se dogodki sporočali na določen časovni interval oz. glede na dogodke v uporabniškem vmesniku. V primeru, da dogodka ne bi bilo mogoče sporočiti na strežnik oz. da od strežnika uporabniški vmesnik ne bi sprejel ustrezne potrditve, bi si uporabniški vmesnik zapomnil, da določenih dogodkov ni uspel uspešno poslati in bi jih poskusil znova poslati kasneje.

Tretja varianta varovanja podatkov tekmovalcev med tekmovanjem bi bila, da bi na uporabniškem vmesniku dodali dodatno programsko logiko za preverjanje delovanja podatkovne shrambe. V primeru izpada sistema v času tekmovanja oz. v primeru, da bi tekmovalec izgubil internetno povezavo med tekmovanjem, bi tekmovalcu na koncu tekmovanja pripravili besedilo, v katero bi zakodirali vse odgovore. Te bi lahko nato tekmovalci poslali po

elektronski pošti, mentorju pa bi jih uvozili v sistem naknadno po končanem tekmovanju (primer sistema v Franciji, več o tem v poglavju 2.2).

Glede na kriterije, navedene v poglavju 1.4.5, z grafov obremenjenosti strežnika med tekmovanjem ne moremo zagotovo trditi, da bi strežnik lahko sprejel 10.000 tekmovalcev naenkrat in da bi s sistemom v tekmovalnem tednu lahko sprejeli tudi do 100.000 tekmovalcev. Realnih preizkušenih ocen za sistem nimamo, ker nismo imeli na voljo takšne količine tekmujočih uporabnikov. Sistem smo testirali z orodjem za testiranje zmogljivosti strežnikov AB (*Apache HTTP server benchmarking tool* [49]) in vsak od virtualnih strežnikov je bil sposoben odgovoriti na 1000 hkratnih zahtevkov, to pomeni, da bi v primeru treh strežnikov sistem potencialno lahko sprejel 3000 hkratnih zahtevkov (v tabeli 5.1 prva vrstica). Če vzamemo za izračun potencialne zmogljivosti strežnika realno pridobljene podatke, kjer vemo, da je 638 hkratnih uporabnikov (Googlova analitika hkratnih uporabnikov na spletni strani, slika 5.11, v tabeli 5.1 druga vrstica) generiralo do maksimalno 142 zahtevkov na sekundo (graf števila zahtevkov na sekundo, slika 5.10, v tabeli 5.1 tretja vrstica), lahko sklepamo, da uporabnik, ki uporablja tekmovalno platformo, naredi povprečno 0,22 zahtevka na sekundo (v tabeli 5.1 četrta vrstica), kar pomeni, da bi lahko z zmogljivostjo 3000 zahtevkov na sekundo uspešno obdelali vsaj 13.636 hkratnih uporabnikov (v tabeli 5.1 peta vrstica), kar je več, kot je številka, navedena v zahtevah delovanja (10.000 tekmovalcev naenkrat). Glede zahteve po zmogljivosti obdelave 100.000 tekmovalcev, porazdeljenih čez petdnevni tekmovalni čas, pa lahko z naslednjimi predpostavkami izračunamo, da ko se tekmovanje izvaja med tednom med poukom, običajno je to čas med 8:00 in 13.00, to pomeni, da je na dan na voljo 5 ur za tekmovanje. Vsako tekmovanje uporabnika traja 45 minut, kar pomeni, da lahko izvedemo na šoli na dan približno 6 tekmovanj uporabnikov. Če imamo lahko hkratnih uporabnikov sistema 13.000 in lahko izvedemo 6 tekmovanj uporabnikov na dan, to pomeni, da jih na dan lahko tekmuje 80.000, v roku 5 dni to pomeni vsaj 400.000 (v tabeli 5.1 šesta vrstica). To pomeni, da smo zahtevo zmogljivosti vseh tekmovalcih krepko presegli.

Vrednost	Izmerjeno	Ocena
Zmogljivost strežnikov v št. hkratnih zahtevkov/s	3000	1500
Št. hkratnih uporabnikov	638	/
Št. hkratnih zahtevkov/s	142	/
Št. zahtevkov na uporabnika/s	0,22	0,22
Skupna zmogljivost hkratnih uporabnikov	13.636	6.818
Dnevna skupna zmogljivost hkratnih uporabnikov	81.816	40.908
5-dnevna zmogljivost uporabnikov	409.080	204.540

Tabela 5.1: Izračun zmogljivosti tekmovalnega sistema Bober, ocena podana z varnostnim faktorjem 2.

Za oceno zmogljivosti sistema v tabeli 5.1, stolpec ,,Ocena“, na strojni opremi opisani v poglavju 5.1, smo uporabili varnostni faktor 2, kar pomeni, da smo zmanjšali zmogljivost streženja zahtevkov sistema na polovico in na podlagi predpostavk ponovno preračunali zmogljivost sistema. Po ocenjeni zmogljivosti z varnostnim faktorjem sistem ne izpolnjuje več zahteve po zmogljivosti hkratnih uporabnikov sistema. Zmogljivost hkratnih uporabnikov z varnostnim faktorjem bi lahko zagotovili tako, da bi povečali število strežnikov, saj bi s tem povečali zmogljivost strežnikov pri hkratnih zahtevkih na sekundo.

Maksimalen čas vzpostavitve sistema v primeru popolnega izpada gručnega sistema smo izmerili tako, da smo vse strežnike ob istem času ponovno zagnali, s tem smo spravili strežnike v nesinhronizirano stanje. Čas do ponovnega stabilnega delovanja ni presegel 20 minut.

Poglavje 6

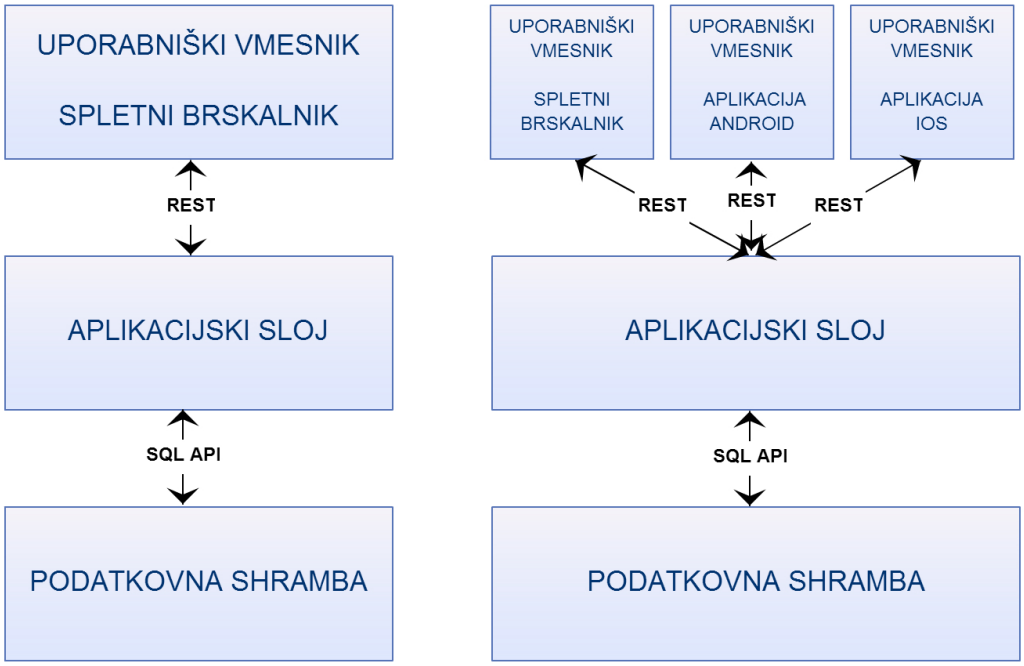
Zaključek in nadaljnje delo

Z izdelavo in izvedbo celotnega sistema smo zadovoljni, implementacija arhitekture s pomočjo treh osnovnih slojev se nam je obrestovala in se izkazala za primerno za uspešno izvedbo projekta. Programski svet tekmovanja Bober v Sloveniji je izvedel anketo med učitelji, kjer smo dobili potrditev, da je bila tehnična izvedba tekmovanja letos veliko bolj tekoča in brez težav kot prejšnja leta, iz tega lahko sklepamo, da je projekt uspešen in da smo naredili korak naprej v zgodovini tekmovanja Bober. Dejstvo pa je, da določene funkcionalnosti na administrativnem vmesniku še niso dokončane, ponekod je uporabniški vmesnik neintuitiven in bi ga bilo mogoče v veliki meri izboljšati, ga narediti prijaznejšega in bolj uporabnega.

Projekt temelji na relacijski podatkovni bazi, kar načeloma pomeni, da je mogoče zmogljivosti povečevati, dokler lahko povečujemo količino pomnilnika v posameznih strežnikih podatkovne shrambe ter dokler ne bodo postali stroški infrastrukture previsoki glede na praktično ničen denarni sklad tekmovanja Bober. Zaradi že trenutno visoke potrebe po pomnilniku bi predlagali, da se tekmovalni del, ki je podvržen veliki količini zahtevkov med tekmovanjem in potrebi po zanesljivem delovanju, prepiše na drugo arhitekturo. Za alternativno arhitekturo bi lahko uporabili tehnologije v oblaku Google App Engine, ki omogočajo praktično neskončno nadgradljiv sistem, seveda za svojo ceno. Vendar pa bi bila ta cena, ker so glavne potrebe po zmo-

gljivosti vsako leto samo med tekmovalnim tednom, praktično nična glede na potrebne vire za izvedbo na arhitekturi, ki je bila sestavljena s pomočjo podatkovne shrambe MySQL Cluster.

Predlog je, da bi za potrebe administracije in obdelave podatkov še vedno uporabljali relacijski model, zasnovan v tem projektu, za del, ki mora povečevati svoje zmogljivosti, pa bi pripravili izvoz podatkov iz obstoječega sistema in jih uvozili na oblačno platformo Google App Engine. Tam bi izvedli tekmovanje, po končanem tekmovanju pa bi podatke/shranjene odgovore tekmovalcev za nadaljnjo obdelavo uvozili nazaj v relacijski podatkovni model.



(a) Trenutna arhitektura sistema Bober.

(b) Predlagana spremenjena arhitektura sistema Bober.

Slika 6.1: 3-slojna arhitektura sistema Bober

Druga varianta razširljivosti tekmovalnega sistema Bober je možnost uporabe mobilnih tehnologij, izvedba tekmovanja na tabličnih računalnikih oz. mobilnih telefonih. Zaradi osnovne trislojne arhitekture in urejene komuni-

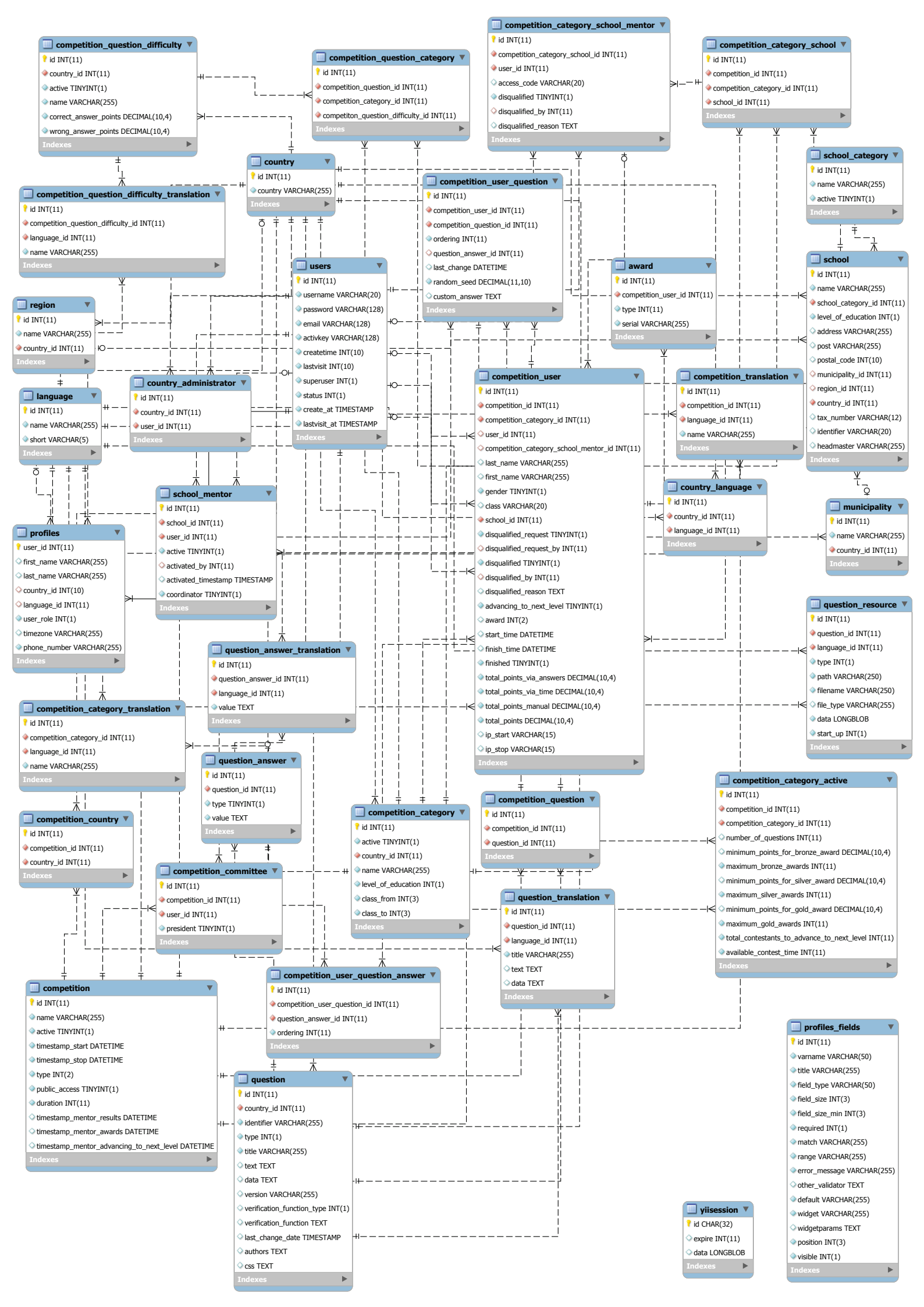
kacije z uporabo REST-klicev med uporabniškim vmesnikom in aplikacijskim slojem bi dodajanje podpore za mobilne naprave dejansko pomenilo implementacijo uporabniškega vmesnika na posamezni mobilni platformi Android, iOS, itd. (trenutna arhitektura na sliki 6.1a poleg predlagane spremenjene arhitekture na sliki 6.1b).

Pomemben pristop, ki smo se ga držali pri izvedbi projekta tekmovalnega sistema, je, da smo v razvoj sistema vključili tudi mlajše generacije študentov Fakultete za računalništvo in informatiko na Univerzi v Ljubljani in s tem poskrbeli, da se bo projekt lahko nadaljeval tudi, ko prvotne razvojne ekipe ne bo več na fakulteti, kar pa je v današnjem svetu bistveno pri vsaki novorazviti tehnologiji in aplikaciji.

V okviru projekta je nastal tudi članek, ki bo objavljen na konferenci ISSEP (*The International Conference on Informatics in Schools: Situation, Evolution and Perspectives*) 2014 [27].

Dodatek A

Podatkovni model sistema Bober


competition_question_difficulty
id INT(11)
country_id INT(11)
active TINYINT(1)
name VARCHAR(255)
correct_answer_points DECIMAL(10,4)
wrong_answer_points DECIMAL(10,4)
Indexes
competition_question_category
id INT(11)
competition_question_id INT(11)
competition_category_id INT(11)
competiton_question_difficulty_id INT(11)
Indexes
competition_category_school_mentor
id INT(11)
competition_category_school_id INT(11)
user_id INT(11)
access_code VARCHAR(20)
disqualified TINYINT(1)
disqualified_by INT(11)
disqualified_reason TEXT
Indexes
competition_category_school
id INT(11)
competition_id INT(11)
competition_category_id INT(11)
school_id INT(11)
Indexes
country
id INT(11)
country VARCHAR(255)
Indexes
competition_user_question
id INT(11)
competition_user_id INT(11)
competition_question_id INT(11)
ordering INT(11)
question_answer_id INT(11)
last_change DATETIME
random_seed DECIMAL(11,10)
custom_answer TEXT
Indexes
school_category
id INT(11)
name VARCHAR(255)
active TINYINT(1)
Indexes
competition_question_difficulty_translation
id INT(11)
competition_question_difficulty_id INT(11)
language_id INT(11)
name VARCHAR(255)
Indexes
users
id INT(11)
username VARCHAR(20)
password VARCHAR(128)
email VARCHAR(128)
activkey VARCHAR(128)
createtime INT(10)
lastvisit INT(10)
superuser INT(1)
status INT(1)
create_at TIMESTAMP
lastvisit_at TIMESTAMP
Indexes
award
id INT(11)
competition_user_id INT(11)
type INT(11)
serial VARCHAR(255)
Indexes
school
id INT(11)
name VARCHAR(255)
school_category_id INT(11)
level_of_education INT(1)
address VARCHAR(255)
post VARCHAR(255)
postal_code INT(10)
municipality_id INT(11)
region_id INT(11)
country_id INT(11)
tax_number VARCHAR(12)
identifier VARCHAR(20)
headmaster VARCHAR(255)
Indexes
region
id INT(11)
name VARCHAR(255)
country_id INT(11)
Indexes
competition_user
id INT(11)
competition_id INT(11)
competition_category_id INT(11)
user_id INT(11)
competition_category_school_mentor_id INT(11)
last_name VARCHAR(255)
first_name VARCHAR(255)
gender TINYINT(1)
class VARCHAR(20)
school_id INT(11)
disqualified_request TINYINT(1)
disqualified_request_by INT(11)
disqualified TINYINT(1)
disqualified_by INT(11)
disqualified_reason TEXT
advancing_to_next_level TINYINT(1)
award INT(2)
start_time DATETIME
finish_time DATETIME
finished TINYINT(1)
total_points_via_answers DECIMAL(10,4)
total_points_via_time DECIMAL(10,4)
total_points_manual DECIMAL(10,4)
total_points DECIMAL(10,4)
ip_start VARCHAR(15)
ip_stop VARCHAR(15)
Indexes
competition_translation
id INT(11)
competition_id INT(11)
language_id INT(11)
name VARCHAR(255)
Indexes
country_administrator
id INT(11)
country_id INT(11)
user_id INT(11)
Indexes
language
id INT(11)
name VARCHAR(255)
short VARCHAR(5)
Indexes
school_mentor
id INT(11)
school_id INT(11)
user_id INT(11)
active TINYINT(1)
activated_by INT(11)
activated_timestamp TIMESTAMP
coordinator TINYINT(1)
Indexes
country_language
id INT(11)
country_id INT(11)
language_id INT(11)
Indexes
municipality
id INT(11)
name VARCHAR(255)
country_id INT(11)
Indexes
profiles
user_id INT(11)
first_name VARCHAR(255)
last_name VARCHAR(255)
country_id INT(10)
language_id INT(11)
user_role INT(1)
timezone VARCHAR(255)
phone_number VARCHAR(255)
Indexes
question_resource
id INT(11)
question_id INT(11)
language_id INT(11)
type INT(1)
path VARCHAR(250)
filename VARCHAR(250)
file_type VARCHAR(255)
data LONGBLOB
start_up INT(1)
Indexes
question_answer_translation
id INT(11)
question_answer_id INT(11)
language_id INT(11)
value TEXT
Indexes
competition_category_translation
id INT(11)
competition_category_id INT(11)
language_id INT(11)
name VARCHAR(255)
Indexes
question_answer
id INT(11)
question_id INT(11)
type TINYINT(1)
value TEXT
Indexes
competition_category_active
id INT(11)
competition_id INT(11)
competition_category_id INT(11)
number_of_questions INT(11)
minimum_points_for_bronze_award DECIMAL(10,4)
maximum_bronze_awards INT(11)
minimum_points_for_silver_award DECIMAL(10,4)
maximum_silver_awards INT(11)
minimum_points_for_gold_award DECIMAL(10,4)
maximum_gold_awards INT(11)
total_contestants_to_advance_to_next_level INT(11)
available_contest_time INT(11)
Indexes
competition_question
id INT(11)
competition_id INT(11)
question_id INT(11)
Indexes
competition_country
id INT(11)
competition_id INT(11)
country_id INT(11)
Indexes
competition_category
id INT(11)
active TINYINT(1)
country_id INT(11)
name VARCHAR(255)
level_of_education INT(1)
class_from INT(3)
class_to INT(3)
Indexes
competition_committee
id INT(11)
competition_id INT(11)
user_id INT(11)
president TINYINT(1)
Indexes
question_translation
id INT(11)
question_id INT(11)
language_id INT(11)
title VARCHAR(255)
text TEXT
data TEXT
Indexes
competition_user_question_answer
id INT(11)
competition_user_question_id INT(11)
question_answer_id INT(11)
ordering INT(11)
Indexes
competition
id INT(11)
name VARCHAR(255)
active TINYINT(1)
timestamp_start DATETIME
timestamp_stop DATETIME
type INT(2)
public_access TINYINT(1)
duration INT(11)
timestamp_mentor_results DATETIME
timestamp_mentor_awards DATETIME
timestamp_mentor_advancing_to_next_level DATETIME
Indexes
question
id INT(11)
country_id INT(11)
identifier VARCHAR(255)
type INT(1)
title VARCHAR(255)
text TEXT
data TEXT
version VARCHAR(255)
verification_function_type INT(1)
verification_function TEXT
last_change_date TIMESTAMP
authors TEXT
css TEXT
Indexes
profiles_fields
id INT(11)
varname VARCHAR(50)
title VARCHAR(255)
field_type VARCHAR(50)
field_size INT(3)
field_size_min INT(3)
required INT(1)
match VARCHAR(255)
range VARCHAR(255)
error_message VARCHAR(255)
other_validator TEXT
default VARCHAR(255)
widget VARCHAR(255)
widgetparams TEXT
position INT(3)
visible INT(1)
Indexes
yiisession
id CHAR(32)
expire INT(11)
data LONGBLOB
Indexes

Literatura

[1] "XAMPP Installers and Downloads for Apache Friends," Apache Friends, Dostopano: 31. 3. 2014. Na voljo na spletu: http://www.apachefriends.org/

[2] "Apache Storm, distributed and fault-tolerant realtime computation," The Apache Software Foundation, Dostopano: 30. 1. 2014. Na voljo na spletu: http://storm.incubator.apache.org/

[3] M. Balodis, "Bebrs Amazon mākonī (shema postavitve tekmovalnega sistema Bober v Amazon oblaku)," Dostopano: 20. 6. 2013. Na voljo na spletu: https://docs.google.com/a/black.si/drawings/d/1uzuPutFsFFjY5WPs-LTnWmldXqYWhd1Kahi8icCr71U/edit

[4] M. Balodis, "Mākonī izvietotas sistēmas slodzes testēšana, optimizācija un mērogošana (Testiranje obremenitve sistema, ki temelji v oblaku, optimizacija in luščenje)," Ph.D. dissertation, Latvijas Universitāte Datorikas Fakultāte, 2012. Na voljo na spletu: https://www.dropbox.com/s/8gvsirw60fg1i7u/korsa_Darbs_Slodzes_Testesana_Martins_Balodis.pdf

[5] M. Balodis, "O tekmovalnem sistemu v Latviji," 2013, zasebna komunikacija.

[6] T. Brisco, "RFC 1794: DNS Support for Load Balancing," IETF, Tech. Rep., 1995, Dostopano: 31. 3. 2014. Na voljo na spletu: http://tools.ietf.org/html/rfc1794

[7] A. Brodnik, "Pravilnik o tekmovanju v informacijski in računalniški pismenosti Bober," 2012, Dostopano: 20. 6. 2013. Na voljo na spletu: http://tekmovanja.acm.si/sites/tekmovanja.acm.si/files/storage/pravilnik_bober.pdf

[8] A. Cartelli, "Castoro (domača spletna stran tekmovanja bober v Italiji)," Dostopano: 2. 5. 2014. Na voljo na spletu: http://www.competenzedigitali.it/castdwn-e.htm

[9] A. Chianis, "In-House Server vs. Cloud: Which Option is Better for Your Business?" Dostopano: 13. 4. 2014. Na voljo na spletu: http://www.businessbee.com/resources/news/technology-buzz/house-server-vs-cloud-option-better-business/

[10] G. Coulouris, J. Dollimore, T. Kindberg, in G. Blair, *Distributed Systems: Concepts and Design*, 5. izdaja. Addison-Wesley, Maj 2011.

[11] Društvo ACM Slovenija, "ACM Tekmovanje - Bober," Dostopano: 1. 2. 2014. Na voljo na spletu: http://tekmovanja.acm.si/bober

[12] "OpeneLMS - open source e-learning for business," e-Learning WMB, Dostopano: 20. 1. 2014. Na voljo na spletu: http://www.openelms.org/

[13] W. Eckerson, *Three Tier Client/Server Architecture: Achieving Scalability, Performance, and Efficiency in Client Server Applications.* Open Information Systems 10, 1995.

[14] "HipHop Virtual Machine for PHP," Facebook, Inc., 2010, Dostopano: 27. 2. 2014. Na voljo na spletu: http://www.hhvm.com/

[15] D. Feenberg, "DNS round robin for web server failover," 2010, Dostopano: 13. 4. 2014. Na voljo na spletu: http://www.nber.org/sys-admin/dns-failover.html

[16] M. Fowler, *Patterns of Enterprise Application Architecture*, 1. izdaja. Addison-Wesley Professional, 2002.

[17] "Google analize," Google, Dostopano: 13. 4. 2014. Na voljo na spletu: http://www.google.com/analytics/

[18] "Google App Engine Quotas," Google, Dostopano: 13. 4. 2014. Na voljo na spletu: https://developers.google.com/appengine/docs/quotas

[19] "MVC Architecture," Google, Dostopano: 31. 3. 2014. Na voljo na spletu: https://developer.chrome.com/apps/app_frameworks

[20] D. Gostiša, "MySQL Cluster postavitev v praksi," Marec 2013, Dostopano: 13. 4. 2014. Na voljo na spletu: http://lusy.fri.uni-lj.si/en/node/138

[21] D. Gostiša in L. Podgoršek, "Navodila za interaktivne naloge Bober," Dostopano: 24. 7. 2013. Na voljo na spletu: http://lusy.fri.uni-lj.si/sl/node/139

[22] M. Hiron, "O tekmovalnem sistemu v Franciji," 2013, zasebna komunikacija.

[23] M. Hiron in R. Naktinis, "Bebras task standard API," Dostopano: 4. 4. 2013. Na voljo na spletu: https://docs.google.com/a/black.si/document/d/1dW9qRu4_-44fe7JagqwiQu7q8_aQpdd5tQXpk4hGKBE/edit

[24] "ILIAS E-Learning - Open Source e-Learning," ILIAS open source e-Learning e.V., Dostopano: 20. 1. 2014. Na voljo na spletu: http://www.ilias.de/

[25] "Intel e-Business Center White Paper, N-tier Architecture Improves Scalability, Availability and Ease of Integration," Intel, Tech. Rep., 2001. Na voljo na spletu: http://cadeiras.iscte.pt/CDSI/fich/N-tier%20Architectures-Intel.pdf

[26] O. Kekäläinen, "Optimizing web server performance with Nginx and PHP," 2013, Dostopano: 25. 3. 2014. Na voljo na spletu: http://seravo.fi/2013/optimizing-web-server-performance-with-nginx-and-php

[27] N. Kristan, D. Gostiša, G. F. Žorž, in A. Brodnik, "A high-availability Bebras Competition System," v *The International Conference on Informatics in Schools: Situation, Evolution and Perspectives*, 2014.

[28] T. Lončarić, A. Vehovec, M. Kastelic, D. Drofenik, S. Divjak, A. Kavčič, M. Marolt, in M. Privošnik, "Slika: Opredelitev porazdeljene podatkovne baze," 2007, Dostopano: 30. 1. 2014. Na voljo na spletu: http://colos1.fri.uni-lj.si/ERI/RACUNALNISTVO/PODATKOVNE_BAZE/opredelitev_porazdeljene_podatkovne_baze.html

[29] "DotLRN - Learn, Research, Network," .LRN Consortium, Dostopano: 20. 1. 2014. Na voljo na spletu: http://dotlrn.org/

[30] K. Matsudaira, "Scalable Web Architecture and Distributed Systems," Dostopano: 30. 1. 2014. Na voljo na spletu: http://aosabook.org/en/distsys.html

[31] "Deployment Patterns," Microsoft, Dostopano: 31. 3. 2014. Na voljo na spletu: http://msdn.microsoft.com/en-us/library/ms998478.aspx

[32] "Moodle.com - We give you powerful free tools to help you educate the world," Moodle Pty Ltd, Dostopano: 20. 1. 2014. Na voljo na spletu: http://moodle.com

[33] "QADPZ - Quite Advanced Distributed Parallel Zystem," The Norwegian University of Science and Technology, Dostopano: 30. 1. 2014. Na voljo na spletu: http://qadpz.sourceforge.net/

[34] "Defining MySQL Cluster Data Nodes," Oracle, Dostopano: 31. 3. 2014. Na voljo na spletu: http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-ndbd-definition.html

[35] "MySQL Cluster Installation and Upgrades," Oracle, Dostopano: 25. 3. 2014. Na voljo na spletu: http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-installation.html

[36] "MySQL Cluster Overview," Oracle, Dostopano: 31. 3. 2014. Na voljo na spletu: http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-overview.html

[37] "MySQL Cluster replication," Oracle, Dostopano: 31. 3. 2014. Na voljo na spletu: https://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-replication.html

[38] "MySQL Cluster Replication Schema and Tables," Oracle, Dostopano: 25. 3. 2014. Na voljo na spletu: http://dev.mysql.com/doc/refman/5.1/en/mysql-cluster-replication-schema.html

[39] "MySQL: SQL Statement Syntax," Oracle, Dostopano: 31. 3. 2014. Na voljo na spletu: https://dev.mysql.com/doc/refman/5.0/en/sql-syntax.html

[40] Oracle, "Guide to Scaling Web Databases with MySQL Cluster," Oracle, Tech. Rep., Junij 2013, Dostopano: 25. 3. 2014. Na voljo na spletu: http://www.mysql.com/why-mysql/white-papers/guide-to-scaling-web-databases-with-mysql-cluster/

[41] "PHP: Hypertext Preprocessor," The PHP Group, 1995, Dostopano: 31. 3. 2014. Na voljo na spletu: http://www.php.net

[42] "PHP: What can PHP do?" The PHP Group, 2005, Dostopano: 31. 3. 2014. Na voljo na spletu: http://www.php.net/manual/en/intro-whatcando.php

[43] B. Pikl, "Navodila za aplikacijo CSV-PDF za generiranje PDF datotek," Dostopano: 13. 4. 2014. Na voljo na spletu: https://lusy.fri.uni-lj.si/redmine/projects/bober/repository/show/CSV-PDF

[44] "Pingdom: Website Monitoring," Pingdom, Dostopano: 13. 4. 2014. Na voljo na spletu: https://www.pingdom.com/

[45] P. J. Sadalage in M. Fowler, *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*, 1. izdaja. Addison-Wesley Professional, 2012.

[46] D. J. Siu, "Google App Engine (GAE) versus Amazon Web Services (AWS)," Dostopano: 13. 4. 2014. Na voljo na spletu: http://notwastingtime.blogspot.com/2010/05/google-app-engine-gae-versus-amazon-web.html

[47] M. Stonebraker, "The Case for Shared Nothing," *Database Engineering, Volume 9, Number 1*, 1986.

[48] D. Sullivan, "PaaS Providers List: 2014 Comparison And Guide," Dostopano: 13. 4. 2014. Na voljo na spletu: http://www.tomsitpro.com/articles/paas-providers,1-1517.html

[49] "AB - Apache HTTP server benchmarking tool," The Apache Software Foundation, Dostopano: 20. 4. 2014. Na voljo na spletu: http://httpd.apache.org/docs/2.2/programs/ab.html

[50] "Joomla! The CMS Trusted By Millions for their Websites," The Joomla Project Team, 2005, Dostopano: 31. 3. 2014. Na voljo na spletu: http://www.joomla.org/

[51] "Bebras, International Contest on Informatics and Computer Fluency," Vilnius University, Dostopano: 20. 12. 2013. Na voljo na spletu: http://www.bebras.org/

[52] "Domača Latvijska spletna stran o tekmovanju Bober," Vilnius University, Dostopano: 24. 12. 2013. Na voljo na spletu: http://www.bebras.lt/

[53] Š. Cerar in J. Demšar, *Bober 2013, naloge in rešitve osnovnošolskega tekmovanja*. Društvo ACM Slovenija, 2013.

[54] WordPress Foundation, "Wordpress Blog," 2003, Dostopano: 31. 3. 2014. Na voljo na spletu: http://wordpress.org/

[55] Yii Software LLC, "Yii PHP Framework: Best for Web 2.0 Development," 2008, Dostopano: 31. 3. 2014. Na voljo na spletu: http://www.yiiframework.com/

[56] "Zabbix :: An Enterprise-Class Open Source Distributed Monitoring Solution," Zabbix SIA, Dostopano: 13. 4. 2014. Na voljo na spletu: http://www.zabbix.com/

[57] B. Škufca, "Apache with mod-php compared to Nginx with php-fpm," 2009, Dostopano: 25. 3. 2014. Na voljo na spletu: http://blog.a2o.si/2009/06/24/apache-mod_php-compared-to-nginx-php-fpm/